2920 SIMULATOR USER'S GUIDE

Manual Order Number: 9800988-02 Rev. B

intel®

2920 SIMULATOR USER'S GUIDE

Manual Order Number: 9800988-02 Rev. B

Copyright © 1979, 1980 Intel Corporation
Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and may be used only to identify Intel products:

BXP	Intellec	Multibus
CREDIT	iSBC	Multimodule
i	iSBX	PROMPT
ICE	Library Manager	Promware
iCS	MCS	RMX
Insite	Megachassis	UPI
Intel	Micromap	μScope
Intelevision		

and the combination of ICE, iCS, iSBC, iSBX, MCS, or RMX and a numerical suffix.

PREFACE

This manual is intended as an introduction to the use of the SM2920 Simulator, the software module which simulates programs written for the Intel 2920 processor chip.

The SM2920 module facilitates the testing and modification of programs for new products designed to utilize the 2920 chip. It does so by

a. letting you use symbolic references for displaying or changing all 2920 registers, flags, and user-named locations in program or data memory

b. tracing, during simulation, the values of items you selected before starting simulation, enabling you to see when and how signals are sampled and handled by your program

In general, the SM2920 module allows you to

- load your object programs from ISIS disk files
- define input signals as functions of time and processor variables or states
- establish halting conditions for each simulation
- establish conditions for trace to be turned on or off and displayed as a table or graph
- create, display, and modify variable values and instructions
- simulate your program, including sampling the input signals as defined above, either one step at a time or in sequence, from zero or other addresses you specify
- save tested or modified code into ISIS disk files

The minimum hardware configuration to run the SM2920 module is as follows:

- INTELLEC or INTELLEC-II with 64K random access memory (RAM)
- Teletypewriter, CRT, or equivalent for console input and output
- Single diskette drive unit

If a line printer is available, it can be used for large-volume or hard-copy output. If an iSBC-310 math board is available, a special version of the SM2920 module (called SM2920.HRD) can use it to speed up simulation. The version SM2920.SFT will run without the math board.

The SM2920 Simulator uses the ISIS-II keyboard and disk input/output functions. You may wish to refer to other documents containing valuable information about this supervisor, the Intellec system, and the other software used on it. These include:

- *Intellec Operator's Manual* 9800129
- *INTELLEC/DOS Diskette Operating System Operator's Manual* 9800206
- *ISIS-II System User's Guide* 9800306

READER'S GUIDE

This book gives a general overview of what the SM2920 module can do, with examples, followed by a detailed discussion of the individual commands you can use, with their full syntax. Examples accompany these individual presentations to familiarize you with the various forms of the commands and certain of their more common uses. The full-session examples can be used as a hands-on introduction to the interactive process of creating a 2920 program and running a simulation session to test it out.

CONTENTS

READER'S GUIDE

CHAPTER 1
INTRODUCTION TO THE SM2920 MODULE

Page

Concepts of Simulation, Testing, and Debugging ... 1-1
Modes of Operation ... 1-1
A Generalized Simulation Session ... 1-1
Symbolic Debugging ... 1-2
TRACE Items ... 1-3
LOADing and SAVing ... 1-3
Input Functions ... 1-4
Display ... 1-4
Breaks in Simulation ... 1-4
SM2920 Simulator Files ... 1-4
HELP ... 1-5
Processor and SM2920 References ... 1-5

CHAPTER 2
DETAILED SYNTAX OF THE INDIVIDUAL COMMANDS

Introduction ... 2-1
Character Set ... 2-2
Tokens ... 2-2
 Keywords ... 2-2
 Reference Keywords ... 2-3
 Command Keywords ... 2-3
 Function Keywords ... 2-3
 User Names ... 2-4
 Symbols ... 2-4
 Numeric Constants ... 2-5
 Special Tokens: Operators, Punctuation ... 2-5
Addresses and Expressions ... 2-6
 Operators and Operands ... 2-6
 Expressions ... 2-7
 Partitions ... 2-7
Notation Used in This Manual ... 2-8

CHAPTER 3
THE SM2920 COMMAND LANGUAGE AND FUNCTIONS

Introduction ... 3-1
Entering Commands at the Console ... 3-3
Utility Commands Involving ISIS-II ... 3-4
 EXIT ... 3-5
 LOAD ... 3-5
 SAVE ... 3-6
 LIST ... 3-6
Number Bases and Radix Commands ... 3-7
 SUFFIX ... 3-7
 BASE ... 3-8
 EVALUATE ... 3-8

Page

Display 2920 Processor and SM2920 Referents ... 3-9
Set 2920 Processor and SM2920 Referents ... 3-9
Displaying or Setting Memory Contents ... 3-10
 Display Memory Contents Command ... 3-10
 Set Memory Contents Command ... 3-11
Symbol Table Commands ... 3-14
Display Symbol Table Command ... 3-15
Define Symbol Command ... 3-16
Change Symbol Command ... 3-16
Remove Symbol Command ... 3-16
Simulation Control Commands ... 3-17
 Specifying the Starting Address ... 3-18
 Halting Conditions in SIMULATE and Set BREAKPOINT Commands ... 3-18
 Set BREAKPOINT Command ... 3-19
 SIMULATE Command ... 3-19
 Display BREAKPOINT Command ... 3-20
 Specify and Display Input Functions ... 3-20
TRACE Control Commands ... 3-21
 CONSOLE Command ... 3-21
 Set and Display TRACE Items Command ... 3-22
 Moving the Buffer Pointer ... 3-22
 Displaying TRACE Data ... 3-23
 MOVE, OLDEST, and NEWEST Commands ... 3-24
 PRINT Command ... 3-24
 Set and Display Qualifier Command ... 3-25
 GRAPHICS Command ... 3-26

CHAPTER 4
A COMPLETE EXAMPLE SIMULATION SESSION

Simulation Speed ... 4-1
Increasing Simulation Speed ... 4-1
Example Session ... 4-3

APPENDIX A
HELP MESSAGES

APPENDIX B
KEYWORDS

APPENDIX C
CONCISE SYNTAX (BNF)

APPENDIX D
GETTING STARTED

APPENDIX E
ERROR MESSAGES

APPENDIX F
LIST OF COMMANDS

APPENDIX G
HEXADECIMAL FORMAT FOR OBJECT FILES

CHAPTER 1
INTRODUCTION TO THE SM2920 MODULE

Concepts of Simulation, Testing, and Debugging

After writing and assembling your 2920 program, you will want to test whether its computations are producing the expected results at the expected time, and outputting them correctly.

In order to check things out, you need to specify inputs, simulate one or more instructions, and be able to stop at a preselected location or condition to review how the simulation is progressing. This last capability is called setting a BREAKPOINT.

You may also want to review the values of key variables or the steps of the simulation as they occurred prior to the break. This is called tracing, i.e., collecting and displaying TRACE information. Similarly, it is useful to be able to display and alter data or instructions in any Random-Access-Memory (RAM) or Read-Only-Memory (ROM) locations, respectively. Normally, RAM is data memory and ROM is program memory.

The 2920 Simulator provides all of the above capabilities. It further facilitates the process of program checkout by letting you use symbolic names instead of numeric addresses. This is an aid to problem-oriented thinking, since you can use the same names in debugging as were used in defining the problem and setting up the original assembler code.

Modes of Operation

The SM2920 module operates in two modes: Interrogation Mode and Simulation Mode.

The *Interrogration Mode* occurs whenever the SM2920 module is not simulating your code. It can be used to prepare the system for a new operation or to investigate the results of the last operation.

The 2920 program residing in memory is simulated during *Simulation Mode.* With the SM2920 capabilities, you may specify a break in simulation on a number of different conditions. Trace data are collected and displayed during simulation.

When a breakpoint is encountered in the Simulation Mode, the SM2920 module reverts to the Interrogration Mode. At this time, you can display the contents of any memory location, processor register, or the trace buffer.

A Generalized Simulation Session

This section describes the main steps in a typical session. You may not always perform all the procedures given here in every simulation session, but the main outline is the same in all sessions. The discussion emphasizes some of the features of the SM2920 module that have not been presented earlier; for details of the command language, see Chapters 2 and 3. The steps in this typical session are:

a. Boot the system and obtain the hyphen (-) prompt from the ISIS-II system. Enter SM2920 command, and obtain the asterisk (*) prompt from the SM2920 module.

b. The SM2920 module has two modes of operation: interrogation and simulation. The asterisk (*) prompt signals that the SM2920 module is in interrogation mode and ready to accept any command.

c. Using the LOAD command, load your program from diskette into simulated ROM.

d. In interrogation mode, prepare the system for simulation by defining symbols and setting simulation breakpoints and trace qualifiers. Also specify what items you would like to see traced.

 The simulation breakpoints and trace qualifiers are SM2920 "pseudo-registers." If the breakpoint register matches, simulation halts; while the trace qualifier matches, trace data collection is enabled and runs whenever simulation runs.

e. Specify the input signals you wish simulated as functions of time or other variables.

f. Enter a SIMULATE command to begin simulation. During simulation the trace data are displayed as they are collected if CONSOLE is ON.

g. When simulation halts, you redisplay the trace data collected during that simulation. SM2920 loads trace data into a trace buffer. Using SM2920 commands, you can position the trace buffer pointer to the information you desire to review, and display one, several, or all the entries in the buffer.

h. When a halt is reached, you can examine and change 2920 memory locations or registers, I/O ports, or SM2920 pseudo-registers, to provide you with valuable information on program operation at the point of termination. You may alter data or register values to examine their effect on the next simulation, or you can patch in changes to your program code itself. You can display and change symbolic values in the symbol table and breakpoint and trace qualifier conditions.

i. Alternate between interrogation and simulation until you have checked everything you want to check.

j. At the end of the simulation session, you can save your debugged code on an ISIS-II diskette file, using the SAVE command. The operation can be specified to save program code and symbol tables.

You can start another session immediately, resetting all parameters to their initial values with a few simple commands, or you can exit to ISIS-II to terminate the session.

This introduction is intended to show you some of the scope and power of the SM2920 module in operation, and to suggest how this design aid can fit into your development cycle.

Symbolic Debugging

The SM2920 software provides symbolic definition of all 2920 registers, flags, memory, and selected pins. The user symbol table is generated along with the object file when a program is assembled using the DEBUG control option. This table may be loaded to Intellec memory for access during Interrogation Mode. You are encouraged to add any additional symbolic values for memory addresses, constants, or variables to this symbol table. Symbols may be substituted for numeric values in any of the SM2920 Simulator commands.

Symbolic reference is a great advantage to the system designer. You are no longer burdened with the need to recall or look up addresses of key locations in your program which can change with each assembly. Meaningful symbols from your source program can be used instead. For example, the command

```
SIMULATE FROM 0 TILL RAM .RESULT > 0.08 AND RAM .VARIABLE32 > 0.875
```

begins execution of the program at location 0. The breakpoint is set to occur when the contents of the RAM variable RESULT exceeds 0.08 and VARIABLE32 in RAM has risen above 0.875. This stops simulation so that you can investigate the status of variables, 2920 processor referents, or SM2920 Simulator referents. You no longer have to be as concerned with the physical location of specific variables. The SM2920 module supplies these addresses automatically from the information stored in the symbol table. In this fashion, any portion of your program can be run continuously or one step at a time. You can specify starting addresses for program execution (using a Set PC command) and break conditions on which the processor should stop execution. This enables you to narrow down the areas and conditions causing unwanted results and then change them.

Trace Items

The SM2920 module can capture valuable trace information while you are executing programs. This data may be displayed either as a table or a graph. This means you can set the names of variables or processor referents that you want traced, and establish the conditions under which trace is activated, such as specific times or specific variable values occurring. The number of trace records that can be kept in the trace buffer is inversely limited by the number of trace items you request. Four hundred is the maximum for the product of number-of-items times number-of-records. If you want ten items recorded for each trace, the maximum number of retained records possible is 40. This restriction has no bearing on trace records sent to a file or other device.

These data provide ample capability for determining how your program was operating prior to the simulation break. It is available for each occurrence of the qualifying conditions that you set up, regardless of whether the break was due to the breakpoint matching or your interrupting simulation via the escape key.

Thus you can run portions of your program until a certain time or signal value occurs, stop automatically, and review the most recent states of key variables. Once you've pinned down areas of probable difficulty, you can step through the code using the SIMULATE ONCE command and display key elements at every point.

As you prove to yourself that each section does exactly and only what you want, you can proceed with certainty in altering or adding new elements or features. In trouble-shooting a changed program, you can enter test values and compare the results with its earlier checked-out operation. Whenever the SM2920 module is not simulating your code, it can be used to prepare the system for a new operation or to investigate the results of the last operation. Data collected during simulation may be routed to a disk file for further analysis by other programs. The format of the data is very easily readable by FORTRAN programs because all trace records are numeric output, in the same floating-point format.

LOADing and SAVing

Programs and symbols are loaded and saved as hexadecimal-format object files. The hexadecimal-format object files produced by the 2920 Assembler (preferably under the DEBUG option) are accepted as program input by the SM2920 module. Program addresses can be referred to symbolically, using the symbols loaded with the object

file, or using symbols created interactively during a simulation session. Program memory can be disassembled into Assembler instructions, or filled with new instructions you enter. The user command language and console interface for the SM2920 module are similar to those used by the new Intel products such as the ICE-85 In-Circuit Emulator.

Input Functions

As mentioned above, the four analog signal inputs to the 2920 chip can be specified, as algebraic combinations of basic functions of time and any other system variables. The simulation of 2920 machine instructions is performed in software. Thus, if an instruction performs an input, the function you specified for that input signal is evaluated. A keyboard-calculator expression-evaluator is also included. The default input and output number bases can be specified.

Display

You may display all 2920 internal registers and memory, and all other related system variables, and modify any of them except those labeled "read-only" in this manual.

Breaks in Simulation

After each instruction is simulated, the breakpoint is consulted to determine whether to stop or continue simulation. You can set up the halting conditions in the breakpoint using any logical combination of relations on any system or user variables.

SM2920 Simulator Files

The SM2920 Simulator consists of three files, SM2920.SFT, SM2920.HRD, and SM2920.OVH. The first two represent different versions of the Simulator: SM2920.HRD provides faster simulation by taking advantage of the iSBC-310 math board, which must be present if this version is used. SM2920.SFT performs the math functions in software. One of these two files should be renamed to SM2920, using ISIS-II, e.g.

```
RENAME :F1:SM2920.SFT to :F1:SM2920
```

The file SM2920.OVH contains the help messages, which allows the Simulator to interactively give information about individual command syntax and options interactively, complementing the data in this manual. The help messages are described below and are reproduced in full in Appendix A. This file should be accessed only by the Simulator.

The Simulator can be invoked by typing the drive and file name after the ISIS prompt. For example:

```
-:F1:SM2920
```

You enter your commands to the Simulator one per line, and terminate each with a carriage-return. You may include comments by preceding them with a semicolon. Commands may be continued by typing an ampersand prior to both the carriage-return and the comment field (if any) of the line to be continued. Characters between the ampersand and the line terminator (carriage-return or line-feed) are ignored, and the ampersand is treated as a space. Each input line may contain no more than 120 characters before the line terminator.

The ESCape key cancels the current activity and returns control of the Simulator to you. This applies to partially typed commands or commands in progress, including a running simulation.

All command keywords can be abbreviated to their first three characters, e.g., SIM for SIMulate or QUA for QUAlifier. Many frequently used command keywords have single character abbreviations, such as T for Time, or S for Simulate. Appendix B shows all keywords and their legal abbreviations.

The EXIT command returns control to ISIS, ending the present use of the SM2920 module:

```
*EXIT
```

HELP

The SM2920 Simulator's first level of help can be summoned by typing HELP after the Simulator gives its asterisk prompt:

```
*HELP
```

You can get more detailed help on specific items by typing the item name after HELP. For example:

```
*HELP SIMULATE
```

A help message about the SIMULATE command will be presented.

You can get a printout of all help messages (see also Appendix A) by typing:

```
*LIST :LP:   ; direct output to line printer
*HELP *      ; asterisk lists all
```

You can gain insight on the use of various processor references by requesting help on cpu$ref and sm$ref. They describe the system variables and how to access them, e.g., TIME, PC, DAR, OVF. Similarly, requesting help on change, display, RAM, and ROM can remind you of how the different forms or references can be used. Suggestions about increasing simulation speed appear in Chapter 4.

Processor and SM2920 References

2920 processor references (labeled 2920 in Table 1-1) represent frequently referenced registers, flags, or bits in the architecture of the processor. References to keywords of the SM2920 module (labeled SM2920) represent numeric valued SM2920 pseudo-registers. You can use these keywords in commands (to display or alter their values), in expressions, or to set test conditions for stopping simulation or trace. Processor and SM2920 reference keywords are shown in Table 1-1; those keywords marked with an asterisk (which is not typed when using them) are for read-only use.

Table 1-1.

Keyword	Range	Discussion
BUFFERSIZE*	0 to 400	Counts records in trace buffer (SM2920)
COUNT*	0 to 2^{32}	Count of instructions simulated since last SIMULATE command
CY	0 ot 1	Carry Flag (2920)
DAR	−1 to +1	Digital/Analog Register (2920)
IN	−VREF to +VREF	Latest sampled-and-held input signal in volts (2920)
IN0, IN1, IN2, IN3	−VREF to +VREF	Analog Inputs in volts (2920)
OUT0*, OUT1*, OUT2*, OUT3*, OUT4*, OUT5*, OUT6*, OUT7*	−VREF to +VREF	Analog output in volts (2920)
OVE	0 or 1	Overflow limiting enable bit (2920)
OVF*	0 or 1	Overflow flag bit (2920)
PC	0 to 191	Program Counter (2920)
SIZE	1 to 192	Number of instructions in program (SM2920)
TIME*	real number	Elapsed time, in seconds (SM2920)
TINST*	real number	Time per instruction, in seconds (SM2920)
TPROG	real number	Time between successive EOPs, in seconds (SM2920)
TQUAL*	real number	Time of last match on qualifier, in seconds (SM2920)
VREF	real number	Reference analog level voltage, in volts. VREF is assumed to be 1.0 volt, though it can be changed. (2920)
XSIZE	4 to 253	Width for trace display; default 79

*read-only

CHAPTER 2
DETAILED SYNTAX OF SM2920 COMMANDS

Introduction

The SM2920 software provides you with an easy-to-use English language command set for controlling SM2920 execution in a variety of functional modes. An example of one complete SM2920 command, in this case one of the forms of the SIMULATE command, is shown in Figure 2-1. This command causes simulation to start and specifies the conditions that will halt simulation. The command is made up of separate tokens or mnemonic codes (character strings): SIMULATE, FROM, 0, TILL, RAM, .X, > , .3, OR, OVF, =, 1. Each of these tokens provides a particular element of information necessary to inform the SM2920 module of the specific command functions to be executed. Table 2-1 defines the function of each of these tokens as well as the match condition and the FROM and TILL clauses. This string of tokens requests SM2920 to start simulation at ROM location 0 and to continue simulation until either RAM location .X is greater than .3 or until the overflow flag is 1. Every SM2920 command is composed of one or more such tokens.

```
SIMULATE FROM 0 TILL RAM .X > .3 OR OVF = 1
```

Figure 2-1. Example of a SIMULATE Command

Table 2-1. Definition of SIMULATE Command Function

Token Number	Name	Function
1	SIMULATE	Simulate command specifier; requests and initiates simulation.
2	FROM	Indicates that the next token or expression is the starting address for simulation.
3	0	Starting address, ROM location zero.
2, 3	FROM clause	FROM 0 causes the program counter (PC) to be loaded with 0, the starting address for the simulation.
4	TILL	Indicates that breakpoint (halting) parameters are to follow.
5	RAM	Indicates that the address lines are to contain the memory address given by the token or numeric expression that follows.
6	.X	The address of the specified memory location.
7	>	Greater than.
8	.3	Constant, three tenths.
5, 6, 7, 8	match-condition	Simulation is to halt if data memory named X becomes greater than .3.
9	OR	Indicates that a second halt parameter is to follow.
10	OVF	Overflow flag.
11	=	Equals.
12	1	Constant, one.
4-12	TILL clause	TILL RAM .X > .3 OR OVF = 1 specifies that the simulation is to halt under either of the conditions given.

Thus, the SM2920 command language is composed of a character set and vocabulary of mnemonic tokens. The character set is used to construct mnemonics and, in turn, the mnemonic tokens are used to construct SM2920 commands.

All of the mnemonics in Figure 2-1 are referred to as tokens. All SM2920 mnemonics are divided into two classes: tokens and special tokens. Tokens encompass constants, keywords, symbols, and user names. Special tokens encompass relational operators, arithmetic operators, and punctuation.

Character Set

The valid characters in the SM2920 command language include upper- and lower-case alphabetic ASCII characters A through Z and the set of digits 0 through 9. The space serves as a delimiter for tokens, and carriage-return/line feed (CRLF) characters are also valid, delimiting command lines. The question mark (?), at sign (@), underline (_), and dollar sign ($) are also valid in user-defined names entered in the command language.

The algebraic operators + and – (binary and unary), asterisk (*), slash (/), relational operators (=, > , <), ampersand (&), semicolon (;), period (.), parentheses [()], and comma (,) constitute the only other valid ASCII characters for the SM2920 Simulator; all other characters are ignored unless occurring within comments.

Alphabetic characters are:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

Numeric characters are:

0123456789 (ABCDEF: hexadecimal characters)

Special characters are:

+ – > = < $'&).,(;*/?@_

This character set is used to construct the vocabulary that constitutes the command language.

The semicolon causes the semicolon and the rest of the current input line to be treated as a comment.

Blank lines are allowed and ignored (CRLF).

Tokens

A token in SM2920 command language is roughly equivalent to a "word" in the English language. It consists of a string of alphanumeric or special characters, possibly augmented by a one-character prefix that serves as a token identifier. Tokens are divided into the following types: keywords, user names, and constants. Examples are: SIMULATE, .START, .X, 3.14159, 0400, 12H.

Keywords

SM2920 recognizes a general class of predefined tokens that are fixed in the command language. They provide three functions. *Reference keywords* are used to specify objects having unique predefined functions. *Command keywords* specify command type. *Function keywords* specify subfunctions within a command. The

following sections define and describe these keyword classes. Appendix B contains a listing of SM2920 keywords and their abbreviations. The reason for discussing the various classes and subsets here is to smooth the later discussions of commands.

Reference Keywords

The command language contains a set of system defined mnemonic tokens that are used to address system objects. Each object such as the PC (Program Counter) is assigned a specific mnemonic that is to be used to address and access the contents of that object. These identifiers are called reference keywords. Reference keywords are used in SM2920 commands to refer to 2920 processor registers and flags, simulation registers, memory locations and blocks, and I/O ports.

The total set of reference keywords is subdivided by types. Each type is referenced by a class name. For example, processor-reference denotes the set of 2920 processor registers. A reference keyword is assigned to each element within the set and is always shown in upper case. For example, DAR denotes the contents of the digital/analog register in the 2920 processor-reference set.

Processor-References and SM2920-References. All of these are assigned reference keywords. (See Table 1-1.)

Simulation Registers. The simulation registers consist of the breakpoint register and the qualification register:

Type	Class Name	Keywords
Breakpoint register	break-reg	BREAKPOINT
Qualification register	qual-reg	QUALIFIER

The breakpoint register is used to halt simulation. The qualification register controls tracing.

Command Keywords

Type	Keywords
Command token	BASE, CONSOLE, DEFINE, EVALUATE, EXIT, HELP, LIST, LOAD, MOVE, NEWEST, OLDEST, PRINT, REMOVE, SAVE, SIMULATE, SUFFIX, TRACE

These tokens are used as command verbs in that they specify the major function (action) to be executed as a result of the command.

Each command keyword is used as the first token of that command. Only one command may be entered on one line and its continuations.

Function Keywords

The function keywords are those tokens that are used as command modifiers. These are used as objects of commands, state or mode specifiers, specifiers of operational controls, and connectors.

Object Keywords

Type	Keywords
Modifier	ALL (print) OFF (console) OLDEST (print) ON (console) SYMBOLS (remove)

These SM2920 keywords are used as command modifiers. For example, in the command "PRINT ALL," the PRINT command is modified to print all of the current contents of the trace buffer, as opposed to a single line, which is the default.

Operational Control Keyword

Type	Keywords
Op Control	FROM (SIMULATE start address) FOREVER (SIMULATE duration) ONCE (SIMULATE duration) TILL (SIMULATE halting conditions)

These are keywords that specify operational conditions required. For example, "SIMULATE FROM 0 TILL RAM.Y < .8" states that simulation shall start from memory location 0 and will continue until the data memory location named Y is less than .8.

Connector Keywords

Type	Keywords
Connector	AND (breakpoint or qualifier registers) OR (breakpoint or qualifier registers)

These keywords connect multiple halting conditions for simulation or qualifying match-conditions for trace. For example, "SIMULATE FROM 0 TILL RAM.X < .8 OR OVF = 0" states that the simulation is to start at zero and to continue until RAM.X becomes less than .8 or the overflow flag becomes zero. Q ∪ A = PC = 0 ANDT > .008 will permit trace records to be collected once per program pass after the first eight milliseconds of simulation.

User Names

The command language permits the user to employ symbolic addressing through the use of user-generated symbols.

Symbols

A symbol is a sequence of contiguous alphanumeric characters, prefixed by a period (.), that references a location in a symbol table. The symbol has two uses. The referenced table location always contains a number; it may be used as an address of an instruction or data value in a program, or it may be used directly as a numerical value (variable). In the first case, the symbol is an alternative method of program addressing (symbolic as opposed to direct numeric addresses). In the second case, it provides a method for storing and retrieving data values symbolically into/from the table itself.

Symbols in the user-program symbol table may be entered, deleted, or referenced through the SM2920 command language. Symbols (names) must conform to translator syntax: a sequence of one or more characters where the first character

must be a letter, an at sign (@), an underline (_), or a question mark (?), and all the characters following may be these or a numeral. However, the SM2920 module truncates all symbols to 31 characters. Thus, two different symbols must be unique in the first 31 characters. In addition, to refer to a symbol in SM2920 command language, you must preface it with a period (.). Intervening spaces are allowed between a period and the symbol following but are not part of the name.

Examples of symbols are:

```
.VAR123 .MSGXIZ .VOLUME .PRESSR .VAR666 .SGNLTO
.@MYFLAG .?WHICHPATH .@WHATTIME?
```

The SM2920 symbol table contains symbols LOADed or DEFINEd, but not yet REMOVEd. When the Intellec system is initialized with the SM2920 module, and a user object program is loaded, the symbols used in that program are loaded into the symbol table if the program was assembled using the DEBUG control option.

You can add symbols to the symbol table or remove any or all symbols currently in memory.

Numeric Constants

A constant is a token that represents a fixed numerical value. The SM2920 module recognizes numeric constants, including floating point constants.

A numeric constant consists of a sequence of numeric characters (digits) and optionally a suffix to specify the constant's radix. A numeric constant has the general form

```
digits [.digits][radix]     ; see explanation below re: bracket notation
```

or

```
.digits[radix]
```

where

digits — is a string of one or more valid numeric digits (0123456789 ABCDEF). Legal digits for a given numeric constant depend upon the radix specified.

radix — denotes the number base or representation for entering and displaying numeric values. The following are the valid SM2920 radix indicators:

Radix	Numeric Base
B	Binary (base 2)
D	Decimal (base 10)
H	Hexadecimal (base 16)

Examples of valid numeric constants:

```
0.10010011B
191D
.0001H
23 (using a radix previously specified or a default)
```

Special Tokens

The command language contains two special token sets that provide special functions: operators and punctuation.

Operators

Type	Class Name	Operators
relational	rel-op	=, <, >, <=, >=
plus	plus-op	+, – (binary and unary)
mult	mult-op	*, /, MOD

Punctuation

Type	Class Name	Punctuation Characters
punctuation	punct-op	' & . , ;) ($ CR LF space

The uses of punctuation characters are defined in those sections that define command formats.

Addresses and Expressions

An address is a construct of numeric valued operands and operators that evaluates to a numeric value.

The SM2920 Simulator evaluates in a left-to-right scan modified by operator precedence for an algebraic sequence in the form:

operand [operator operand] . . .

Operators

Operators are used in expressions and in commands. A summary of SM2920 operators is shown below. The binary (arithmetic) operators are listed in their group order of precedence from highest precedence to lowest, i.e., MOD, *, and / have equal precedence higher than + or –, which have equal precedence. MASK has lowest precedence. When several operators of the same precedence appear in an expression, they are evaluated left to right. Expressions in parentheses are evaluated first.

Type	Operator	Interpretation	
Precedence	()	Controls order of evaluation	
Binary (Arithmetic)	MOD	Modulus remainder	same precedence
	*	Multiplication	
	/	Division	
	+	Addition	same precedence
	–	Subtraction	
	MASK	Bitwise AND	
Unary-op	+	Single positive quantity	
	–	Negative quantity	

Operands

Operands are numeric values, and have the general forms shown below. These are the "primaries," which are allowed as restricted expressions in later discussions of SM2920 commands.

[unary-op]	(expression) numeric constant symbol processor-reference SM2920-reference constant-name, e.g., PI, ONE, HPI, TPI function (expr), e.g., SIN (TPI*T), SQR(T), SQW(T), EXP(T), COS(T), LOG(T), SAW (T)

where

(exp) is an operand whose value is the value of the parenthesized expression, e.g., (1 + 2 + 3) = 6 (operand value).

Numeric command parameters that implicitly refer to locations in memory must be addresses. Examples of addresses are:

0AFH
PC + 10
PC + 3*(RAM.FILTER MOD 7)
(RAM PC + 10) MASK F0H

Expressions

An expression can be either

address: expression value is address value

or

RAM address: contents of RAM location

Numeric command parameters that do not implicitly refer to locations in memory can be expressions in either form.

Examples of expressions are:

RAM .ARRAY + 3

PC + 3*(RAM .FILTER MOD 1/3)

NOTE

An expression of the form "RAM address" must be enclosed in parentheses if arithmetic operations are to be performed on its value.

Partitions

Partition refers to an address or range of contiguous addresses and is used in defining certain commands, e.g., the SAVE command. There are several forms in which you can specify a partition:

address
address TO address
address LENGTH address

The third form is equivalent to *first address* TO *first address + second address – 1.*

Notation Used in This Manual

This manual employs a set of notational symbols and conventions to describe the structure of commands and other language constructs. Items 1 and 2 below specify the use of terms and tokens, respectively. The features of this notation system are as follows:

1. A lower-case entry in the description of a command will stand for a set or class of tokens. To create an actual operable command, you must enter a particular member of this class. For example, the lower-case entry

 input-channel

 means that the command will accept any of four tokens: IN0, IN1, IN2, or IN3.

 Another example:

 shiftcode

 would mean any of the codes from L02, L01, R00, R01, . . . , R13.

 Each class-name given in the syntax description of a command is explained in the discussion of semantics that accompanies the command.

2. An upper-case entry is a token that must be used literally as given. A valid abbreviation of that token may substitute for the full token as given. The token may be a command word, or it may be a particular member of a class of references. For example, the upper-case entry

 DEFINE

 is a command word that must be used as given unless abbreviated. The abbreviation DEF may be used in place of DEFINE. As another example, the upper-case entry

 R02

 means this shiftcode must be named as and where given.

3. A single required entry is shown without any enclosures, whereas free options or choices among restricted alternatives are specially denoted. For example, in the following command syntax

 PRINT [ALL]

 the token PRINT is required. The significance of the brackets around ALL is explained in (5) below.

4. Where only one entry must be selected from a menu of two or more items, the choices for the required entry are given in a vertical arrangement enclosed by two vertical lines. For example

   ```
   SIMULATE |TILL break|
            |FOREVER   |
            |ONCE      |
   ```

 indicates that one of these three choices must be selected; the token SIMULATE is required as given.

5. An optional entry is enclosed in brackets []. Where a choice exists for an optional entry, the choices are given in a vertical arrangement enclosed in brackets. For example, in the command

 PRINT [ALL]

 the entry ALL is optional. Omitting an optional entry usually alters the operation of the command, as explained under each command description.

6. A group of inclusive choices is given in vertical arrangement and enclosed in braces { } . "Inclusive" means that more than one of the items can be entered in the same command, and items can appear in any order; no item can be used twice. The menu of inclusive items represents an optional entry or entries. For example:

 LOAD pathname { NOCODE / NOSYMBOL }

 This notation indicates that none, one, or both choices of NOCODE or NOSYMBOL may be included in one command; if more than one is used, the entries can be in any order.

 To complete the example,

 LOAD f1:MYPROG.HEX NOSYMBOL NOCODE

 is a valid command.

7. Where two or more choices are mutually exclusive, and as a group they constitute one choice within an arrangement of inclusive entries, the mutually exclusive items are given on one line separated by slashes (/). For example:

 SAVE pathname { NOCODE/partition / NOSYMBOLS }

 This example indicates that the optional entries are the items NOSYMBOLS, and either NOCODE or memory partition; the entries can appear in any order. NOCODE and partition are mutually exclusive.

8. Where mutually exclusive entries can be shown on one line, the following shorthand notation can be used:

 SUFFIX = B/D/H

 This example is equivalent to

 SUFFIX = | B |
 | D |
 | H |

9. Where an entry can be repeated indefinitely at the user's option, the syntax is notated by enclosing the repeatable entry in brackets [] followed by three periods (. . .). For example,

 operand [operator operand] . . .

 indicates that operator operand can be repeated as many times as desired within system input line length restrictions, etc.

 In addition, some repeatable entries require a particular item separator to precede each repeat after the first entry. When an item separator is required, it is also enclosed in the brackets, preceding the repeatable entry.

CHAPTER 3
THE SM2920 COMMAND LANGUAGE AND SM2920 FUNCTIONS

Introduction

Chapter 3 contains discussions, examples, and syntax summaries for each of the SM2920 commands. The chapter begins with a summary of information on entering commands at the console. The commands have been classified into sections for discussion; commands in a given section are related in function.

The following brief outline of Chapter 3 shows how the SM2920 commands have been classified for discussion. (HELP appears in Chapter 1 and Appendix A.)

Utility Commands Involving ISIS-II

- EXIT Command
- LOAD Command
- SAVE Command
- LIST Command

Number Bases and Radix Commands

- SUFFIX Command
- BASE Command
- EVALUATE Command

Set and Display 2920 Processor and SM2920 References

- Display Processor and SM2920 References
- Set Processor and SM2920 References

Memory Content Commands

- Display Memory Contents Command
- Set Memory Contents Command

Symbol Table Command

- Display Symbol Table Command
- DEFINE Symbol Command
- Change Symbol Command
- REMOVE Symbol Command

Simulation Control Commands

- SIMULATE Command
- Set Breakpoint Register Command
- Display Breakpoint Command
- Specify and Display Input Functions Command

Trace Control Commands

- CONSOLE Command
- Set Trace Items Command
- MOVE, OLDEST, and NEWEST Commands
- PRINT Command
- Set Qualifier Register Command
- Display Trace Items Command
- Display Qualifier
- GRAPHICS Command

Table 3-1 lists the commands separately in alphabetical order by command title. You can use this table to locate any command or command function in this manual.

Table 3-1. Alphabetical Summary of SM2920 Commands

Command Title	Operations Included In Command	Manual Reference
Change Symbol	Change address or variable value corresponding to a previously defined symbol.	Symbol Table Commands
CONSOLE	Set and display CONSOLE on/off.	Trace Commands
DEFINE Symbol	Define a symbol and assign it a corresponding value (typically an address).	Symbol Table Commands
Display Input Function	Display current settings of input functions per channel desired.	Simulation Control Commands
Display Breakpoint Register	Display the current setting of the breakpoint register.	Simulation Control Commands
Display Memory Contents	Display contents of one or more memory addresses.	Memory Contents Commands
Display Processor or SM2920 Reference	Display contents of processor or SM2920 references.	Set and Display Processor or SM2920 Register Commands
Display Radix	Display current input or output radix.	Number Bases and Radix Commands
Display Symbol Table	Display address or value from symbol table corresponding to a user-defined symbol.	Symbol Table Commands
Display Trace Items	TRACE	Trace Control Commands
Display Qualifier Register	Display qualification register setting.	Trace Control Commands
EVALUATE	Evaluate and display the expression entered, in the three bases B, D, H.	Number Base and Radix Commands
EXIT	Terminate simulation session and return control to ISIS.	Utility Commands Involving ISIS-II
GRAPHICS	Display trace data as graph (if ON) or table (if OFF).	Trace Control Commands
LIST	Send SM2920 console output to designated pathname also.	Utility Commands Involving ISIS-II
LOAD	Load user program into simulated ROM.	Utility Commands Involving ISIS-II
MOVE, OLDEST, NEWEST	Move trace buffer pointer to desired position.	Trace Control Commands
PRINT	Display trace data.	Trace Control Commands
REMOVE Symbol	Delete one or more user-defined symbols from the SM2920 symbol table.	Symbol Table Commands
SAVE	Save user program on pathname.	Utility Commands Involving ISIS-II
Set Breakpoint Register	Set breakpoint register to condition for halting simulation.	Simulation Control Commands
Specify Input Function	Specify input functions per channel desired.	Simulation Control Commands
Set Memory Contents	Set the contents of one or more memory locations.	Memory Contents Commands
Set Processor Reference	Set contents of processor or SM2920 reference.	Set and Display Processor or SM2920 Register Commands
Set Qualification Register	Set qualification register to condition for enabling trace data collection.	Trace Control Commands
Set Radix	Set default input or output radix.	Number Bases and Radix Commands
Set TRACE Items	Specify items to be collected as trace data.	Trace Control Commands
SIMULATE	Begin simulation, accepting default or current halting and starting conditions, or specify new ones.	Simulation Control Commands

Entering Commands at the Console

The SM2920 Simulator displays an asterisk prompt (*) at the left margin when it is ready to accept a command from the console.

Each command is entered as a command line. The command line consists of one or more input lines; the length of an input line is limited by ISIS-II to 122 characters, of which the last two are always the carriage-return/line feed pair.

SM2920 recognizes the carriage return as the terminator for a command line. If it is necessary to use more than one input line to enter a command, each intermediate input line should end with an ampersand (&), not in a comment. When SM2920 encounters the ampersand, it suppresses the interpretation of the command that would occur on encountering the carriage return that follows. After the carriage return is executed, SM2920 displays a double asterisk prompt (**) to acknowledge the continuation of the command line.

Tokens in the command are separated by blanks unless the construct requires another form of separator. For example, tokens in a list are separated by commas; in this case, blanks may be inserted for clarity but are not required.

Any input line may include comments. A semicolon (;) precedes the comments. The comments must appear after any portion of the command that is on that input line; in other words, if the first character in an input line is a semicolon (;), the entire input line must consist of comments. Characters in a comment are not interpreted by the SM2920 module, and are not stored internally. The main use of comments is to document a simulation session while it is in progress. This is particularly useful when running SM2920 under the ISIS-II SUBMIT facility.

Comments may not be continued from input line to input line. If an ampersand is used to continue a command line that also contains comments, the ampersand must come before the comment. An ampersand that is embedded in a comment is ignored by the SM2920 Simulator.

You can use ISIS-II editing capabilities to correct errors in the current input line. The line-editing characters are as follows:

Characters	Result
RUBOUT	Delete last character entered in input line. The deleted character is echoed immediately. The RUBOUT function can be repeated, deleting one character each time it is pressed.
CTRL X	Delete entire input line.
CTRL R	Display entire input line as entered so far. This is useful after a RUBOUT, to review which characters have been deleted.
ESC	Cancel entire command being entered (or executed).
CTRL P	Input next character literally.
Carriage Return	Terminate input line or command line.
Line Feed	Terminate input line or command line.

Once a line terminator (carriage return or line feed) has been entered, that line can no longer be edited.

Within symbol names or numeric constants, the dollar sign ($) is ignored by the SM2920 module. You can use it as a separator when you want to combine words into one user name. For example, suppose you wanted to combine DATA and STS. Instead of DATAANDSTS, you can use the $ character as a separator: DATAANDSTS. (The $ is not saved in the symbol table.)

Utility Commands Involving ISIS-II

The Intel Systems Implementation Supervisor (ISIS-II) is the diskette operating system for the Intellec Microcomputer Development System. The SM2920 Simulator runs under ISIS-II control, and can call upon ISIS-II for file management functions. To execute the Simulator, enter SM2920 after an ISIS-II prompt character (hyphen).

The following commands are included in this section: EXIT, LOAD, SAVE, and LIST. With the exception of EXIT, these commands are used to reference files or devices via ISIS-II pathnames. For diskette files, the format of pathname is as follows (a pathname may not contain blanks):

:drive:filename

The entry :drive: stands for one of the references to ISIS diskette drives. See the ISIS manual for futher information.

The entry filename must follow the second colon (after drive) without any intervening spaces. A filename has the following components:

identifier [.extension]

The entry identifier is a name assigned by the user, and is made up of one to six alphanumeric characters. The extension is an optional part of the filename, consisting of one to three alphanumeric characters preceded by a single period. The extension must be used if it is present in the directory listing of the file on the diskette. If used, the extension follows the identifier without any spaces. Some extensions (e.g., .BAK, .LST) are assigned by system processors; others can be assigned at the desire of the user. An extension provides a second level of file identification; it can be used to identify different versions of the same program, or to give supplemental information about the file (e.g., author, data, version).

Assembled programs ready to run (simulate) may have the default system-assigned extension .HEX, although they may have extensions assigned by the user.

For devices other than diskette files, the format of pathname is as follows:

:device:

The following devices are commonly accessed in SM2920 Simulator commands:

:Device:	Output Device
:LP:	Line printer
:HP:	High-speed tape punch
:TO:	Teletypewriter printer
:CO:	Console display
:CI:	Console input
:HR:	Paper tape reader

For more information on ISIS-II filenames and device codes, refer to the *ISIS-II System User's Guide.*

The EXIT command ends the simulation session and returns control to ISIS-II.

The LOAD command loads the hexadecimal object code from the named file into simulated ROM memory, and optionally loads the program symbol table into the symbol table. The symbols are placed in reserved areas of Intellec memory. Symbols are loaded in the table in the order that they appear in the file.

The LOAD command can include one or more modifiers to control what is to be loaded. If NOCODE is included, the program code is omitted from the load. If NOSYMBOL is included, the program symbol table is not loaded. Any combination of one or two modifiers may be included, although the command with both modifiers represents a "null" command. No modifier may be named twice in the same LOAD command.

The SAVE command copies the entire user program and symbols (or the partition explicitly specified) onto the specified pathname. If a diskette is referenced, and the diskette installed on the given drive has the named file in its directory, ISIS-II deletes its directory entry and overwrites it.

The SAVE command can include one or more modifiers to control what is to be saved. If NOSYMBOL is included, the symbol table is not SAVEd. The modifiers NOCODE and partition are mutually exclusive: if one is used, the other may not be included. If NOCODE is included, the program object code is not SAVEd. If partition is included, only the code stored in the memory address in the partition (range of addresses) is saved.

When more than one modifier is used with SAVE, separate them with spaces. No modifier may be used twice in the same SAVE command. The SAVE operation does not alter the program code or symbol table.

The LIST command saves a record of the SM2920 session console input and output, including high-volume data such as trace data, on a hard-copy device or on a diskette file. Using the GRAPHICS feature described under TRACE commands, you can create a line-printer graph of chosen variables or expressions for simulations of any length. Only one LIST device or file (other than the console) can be specified (active) at a given time. Devices that can be specified, if present, are a line printer (:LP:), high-speed paper tape punch (:HP:), and teletypewriter printer (:TO:).

Instead of a hard-copy device, a diskette file can be specified. If the output is to a diskette file, the file is opened when the LIST command is invoked. If a file for that name already exists, it is deleted.

When LIST is in effect (with a device or file other than :CO:), all commands input (including comments) and all commands output from the SM2920 Simulator (including system prompts, commands, and error messages) are sent both to the named device or file and to the console display.

To restore output to the console only (no other device), use the command LIST:CO:.

EXIT Command

The EXIT command keyword returns control from the SM2920 module to ISIS-II.

Example:

```
EXIT
```

LOAD Command

LOAD pathname $\left\{\begin{matrix}\text{NOCODE} \\ \text{NOSYMBOL}\end{matrix}\right\}$

where

LOAD — is a command keyword that loads the software on the given file and drive into the simulated 2920 memory. LOAD sets SIZE (number of instructions in program) automatically.

pathname	is the ISIS-II pathname of device or file.
NOCODE	is a modifier specifying that program code is not to be loaded.
NOSYMBOL	is a modifier specifying that the program symbol table is not to be loaded.

Examples:

```
LOAD :F0:TEST.VR1
LOAD :F1:MYPROG.HEX
LOAD :F2:COUNT.ONE NOCODE
LOAD :F3:NEWCOD NOSYMBOL
```

SAVE Command

SAVE pathname {NOSYMBOL / NOCODE/partition}

where

SAVE	is the command keyword that directs the SM2920 module to write the designated software elements to the indicated file or device.
pathname	is the name of the device or file that is to receive the saved information. If a diskette file of the same name already exists, it will be deleted.
NOCODE	is a modifier specifying that the program code is not to be saved.
partition	is a construct specifying a range of one or more contiguous locations in memory; the contents of the specified locations are saved, but code in other locations is not copied. (Refer to the discussion under the Memory Contents Commands for the forms used to specify memory partitions.) If neither partition nor NOCODE is specified, then the range of memory last loaded or saved will be saved.
NOSYMBOL	is a modifier specifying that the symbol table is not to be saved to diskette.

Examples:

```
SAVE :F1:TEST.HEX
SAVE :F0:MYPROG 80 TO 160
SAVE :F2:COUNT.TWO NOSYMBOL
SAVE :F3:NEWSYM NOCODE
```

LIST Command

LIST pathname

where

LIST	is the command keyword directing all SM2920 console input and output to also be copied to the specified device or file.
pathname	is an ISIS-II device or file name indicating the destination of the output. If there is an existing diskette file of the same name on the specified diskette, it will be deleted and overwritten.

Examples:

```
LIST :LP:
LIST :CO:
LIST :F1:SIMFIL.1
```

Number Bases and Radix Commands

SM2920 commands and displays involve several different number bases (radixes). This section describes the various radixes used by the Simulator and the commands used to control some of them. A few interactions involve radixes set by the Simulator and cannot be changed, but most are under your control.

The SUFFIX and BASE commands given in detail in this section refer to the radixes used for console input and console output. The EVALUATE command provides the capability to evaluate an expression and display the result in all bases.

SUFFIX Command

```
SUFFIX
SUFFIX = |B|
         |D|
         |H|
```

where

SUFFIX	is a command keyword referring to the implicit console input radix. The token SUFFIX alone displays the current setting (B, D, or H).
=	is the assignment operator, indicating that the new setting is to follow.
B	is binary radix (base 2).
D	is decimal radix (base 10).
H	is hexadecimal radix (base 16).

The implicit input radix is the base used by SM2920 to interpret numbers entered from the console without an explicit radix. The initial implicit radix is decimal (D).

To display the current implicit input radix, enter the command token SUFFIX followed by a carriage return. The implicit input radix can be B, D, or H, as defined earlier.

You can change the implicit input radix by entering a command with the form SUFFIX = x, where x is any of the characters B, D, or H. This SUFFIX command can be used where several numbers are to be entered in the same radix.

An explicit radix always takes precedence over the implicit radix. If the digits in the number entered cannot be interpreted in either the explicit radix, if present, or the implicit radix, an error message is displayed.

Examples:

SUFFIX	(display current suffix)
SUFFIX = B	(set suffix to binary)

BASE Command

```
BASE
BASE = |B|
       |D|
       |H|
```

where

BASE	is a command keyword referring to the system console output radix. The token BASE alone displays the current setting (B, D, or H).
=	is the assignment operator, indicating that the new setting is to follow.
B	is binary radix (base 2).
D	is decimal radix (base 10).
H	is hexadecimal radix (base 16).

To display the current console output radix, enter the command token BASE followed by a carriage return. The display consists of a single character, B, D, or H.

The initial output radix is decimal. You can change the console output radix by entering a command with the form BASE = x, where x is one of the single characters, B, D, or H. Once the radix is set with a BASE command, it stays in effect until another BASE command is entered. (However, the EVALUATE command always displays in all three bases. Also, numbers outside the range $-1.0 \leqslant X \leqslant +1.0$ are always displayed in decimal scientific notation.)

The BASE affects all numeric output, in particular trace data display. Since numeric displays in the various radixes require different numbers of characters, the number of columns of trace data (i.e., the number of trace items specified) varies, as to how much can fit on the console screen. However, changing the BASE does not affect the trace buffer contents, so you can examine trace data in all three radixes.

Examples:

BASE	(display current output radix)
BASE = B	(set output radix to binary)

EVALUATE Command

```
EVALUATE expression [SYMBOLICALLY]
```

where

EVALUATE	is a command keyword that directs the SM2920 module to evaluate the expression and display the result in all three number bases.

The EVALUATE command evaluates the expression and displays the result in all three BASEs. This command has the form EVALUATE x, where x is any expression. Upon receiving this command, the SM2920 module evaluates the expression to a single number, and displays on a single line the result in the three bases B, D, and H. If SYMBOLICALLY is used, the display is the symbol whose value is closest to (and less than) that of the expression.

Display 2920 Processor and SM2920 Referents

To display the current contents of any 2920 processor or SM2920 referent, enter the token for the desired item followed by a carriage return. The current setting is displayed on the next line in the current output radix.

Examples:

```
PC
PC = 191
CY
CY = 1
DAR
DAR = 0.10101010100000000000000
```

Table 1-1 lists the 2920 processor and SM2920 referents.

Set 2920 Processor and SM2920 Referents

To change the contents of a 2920 processor or SM2920 referent, use a command of the form:

```
register = expression
```

where

register	is the keyword name of any of the writeable (not read-only) processor or SM2920 referents listed in Table 1-1.
=	is the assignment operator.
expression	is as described in Chapter 2.

Each of the registers that can be changed with this command has a legal range of values. If the new contents represent a value outside this legal range, the SM2920 Simulator issues an error message and no change occurs. However, when this out-of-range condition occurs in an attempt to change the RAM or the DAR, a warning is issued and the location is set to the closest legal value.

Examples:

```
OVE = 1
PC = 80
PC = PC + 1
DAR = .101101011B     ;
IN = 1/2              ;
SIZE = 96             ; instructions in program
TPROG = 1/13020       ; time for one program pass
XSIZE = 79            ; proper screen width for Intel terminals (80 causes
                      ; skipped lines)--for printing, set XSI to last printing col-
                      ; umn, e.g., 132 for many line-printers. XSI controls trace
                      ; data display width for both graphs and tables.
```

The IN0, IN1, IN2, IN3 keywords are not normally set in commands like those above because they are usually functions of time rather than constants (see page 3-20).

Displaying or Setting Memory Contents

The form of the various commands discussed in this section is shown below. A detailed discussion follows, with examples.

Display Memory Contents Command

RAM partition

or

ROM partition

where

RAM	means the contents of a data memory location.
ROM	means the contents of a 2920 program memory location.
partition	means one or more contiguous memory locations.

As described earlier, a partition can be a single address or one of the following types of constructs:

address TO address

address LENGTH address

To display the contents of one or more locations in memory, enter the appropriate memory-content-reference followed by a carriage return. Later in this section we discuss the ways to refer to a range of addresses. Here are some examples of display commands that involve single addresses using ROM and RAM. For these examples, assume these equalities and conditions:

.AA = 10D (symbolic reference),
RAM location 10D contains .5 ROM 10D contains ADD .OSC,KP1,R02
RAM location 11D contains .6 ROM 11D contains SUB .NXT,.OSC,IN3

The commands are shown below as they would be entered. (The terminating carriage return is not shown.) The content displayed for each command is shown on the next line as displayed by the SM2920 module.

```
RAM 10                    ROM 10
RAM 10 = .5               ROM 010 = ADD  .OSC,KP1,R02

RAM .AA                   ROM .AA
RAM 10 = .5               ROM 010 = ADD  .OSC,KP1,R02

RAM .AA + 1               ROM .AA + 1
RAM 11 = .6               ROM 011 = SUB  .NXT,.OSC,IN3
```

To display the contents of a range of addresses, use the form

RAM partition

or

-ROM partition

The first form of partition uses the keyword TO. The address on the right of the keyword TO must be greater than the one on the left.

The second form of partition shown uses the keyword LENGTH. The address preceding that keyword is the starting address in the range. The address or constant following LENGTH gives the number of addresses.

Here is an example, using memory content references to display the contents of a range of addresses. The following could represent a section of data memory displayed in hexadecimal radix:

```
*RAM 10D TO 26D
RAM 10 = 0.224C4F
RAM 11 = 0.200A0D
RAM 12 = 0.202020
RAM 13 = 0.202020
RAM 14 = 0.532020
RAM 15 = 0.6F626D
RAM 16 = 0.5F6369
RAM 17 = 0.656665
RAM 18 = 0.636E65
RAM 19 = 0.0A0D20
RAM 20 = 0.202020
RAM 21 = 0.202020
RAM 22 = 0.20202E
RAM 23 = 0.0A0D0A
RAM 24 = 0.202020
RAM 25 = 0.736944
RAM 26 = 0.79616C
```

Set Memory Content Command

RAM partition = new-RAM-content [,new-RAM-content] . . .

or

ROM partition = new-ROM-content

where

partition	is one or more contiguous locations in memory.
=	is the assignment operator.
new-RAM-content	is one of the following types of entries, to be used as the new-contents: expression RAM partition
new-ROM-content	is one of the following: instruction ROM partition

A command of the form RAM .X = RAM.Y copies the contents of address Y to become the contents of X also.

The commands used to set a range of addresses differ in some details. One way to set a range of addresses is with a command of the form:

RAM address = list of new-content values (separated by commas)

or

ROM address = ROM partition

In this context, each new-content can be

expression	representing a single numeric new-content.
RAM partition	is a form of new-content allowing you to copy a location or a sequence of locations from one address or partition to another.

With this form, the address on the left of the equal sign gives the starting location, and the number of new-contents in the list to the right of the equal sign tells the SM2920 module how many consecutive addresses to set. Consecutive locations starting with the one given are changed to the new-contents in the list, in the left-to-right order. Here are some examples showing the use of this form of the Set Memory and Display Memory command:

```
*
*RAM 10 = 1/3, 2/5, 3/7
*
*RAM 10 to 12
RAM 10 = 0.33333333
RAM 11 = 0.39999998
RAM 12 = 0.42857139
*
*ROM 10 LEN 3
ROM 010 = LDA  .P031,.P032,R00,CVT7
ROM 011 = LDA  .P032,.P030,R00,NOP
ROM 012 = SUB  .P303,.P030,L02,CVT6
*
*ROM 10 = ROM 12
*ROM 10
ROM 010 = SUB  .P030,.P030,L02,CVT6
*ROM 100 = ROM 5 LEN 3          ; sets ROM 100, 101, 102 to instructions
                                ; from ROM 5,6,7
```

The new-contents ROM 5 LEN 3 is treated as a series of quantities. New-contents of this form can be used to perform block moves of memory. Blocks must not overlap. For example, to insert a new instruction,

```
*ROM 100 = ROM.PATCH LEN 50
*ROM.PATCH = LDA .X,.X,R1
*ROM.PATCH + 1 = ROM 100 LEN 50
```

To set a range of addresses all to the same new new-content, use a command of the form:

RAM partition = expression

or

ROM partition = instruction

The forms of partition are discussed above in this section. All addresses in the partition are set to the single new new-content. The following example shows some of the possible results with this command form:

RAM 0 TO 39 = ONE

or

ROM 0 TO 191 = NOP

To assign a new content to a single address, use a command with the form:

RAM address = expression

or

ROM address = instruction

The terms "address" and "expression" represent the types of entries discussed above in Chapter 2. An instruction is any legal assembler instruction as discussed in the 2920 Assembler Manual. However, all symbol references must be predefined and preceded by a period, as with all Simulator symbolic references, and absolute addresses may be used as source or destination operands.

The legal values for the data replacement expression must be $-1.0 <$ exp value $< +1.0$. Out-of-range items force the use of the nearest limit, e.g., $+1.0$ or -1.0.

Here are some examples of setting memory contents. The first line of each example shows the command that sets the new-contents. The second line gives a command that produces a display of the contents just set, and the third line shows the resulting display.

```
*RAM .VAR33 = 1/2 * 13/24
*RAM .VAR33
RAM .VAR33 = 0.27083333
*RAM 10 = 1/4
*RAM 10
RAM 10 = 0.25000000
*ROM 010 = ADD .P031, .P032, R00, CVT6
*ROM 010
ROM 010 = ADD .P031, .P032, R00, CVT6
*ROM 10 = LDA .P031, .P032, CVT7
*ROM 10
ROM 010 = LDA .P031, .P032, R00, CVT7
```

The last form of the Set Memory command sets the new-contents of each address in a range partition to the corresponding new-content in a list of new-contents. This form is as follows:

RAM partition = list of new-contents

or

ROM partition = ROM partition

where new-content is described above.

If the number of addresses in the target range to be changed is greater than the number of new-contents in the list or partition, then the addresses are filled with the new-contents in left-to-right order, repeating the list or partition as necessary to fill all the locations.

If the number of new-contents is greater than the number of addresses in the partition, the lowest address in the range receives the first new-content, and successive addresses in the range receive new-contents in left-to-right order until all locations in the range have received new-contents. The excess new-contents are then detected by the SM2920 module as an error condition, and an error message is displayed. The excess new-contents are lost.

Here are a few examples showing this form of command:

```
*RAM 15 TO 17 = .4, .5, .75          ; sets 15, 16, 17
*ROM 100 TO 102 = ROM 5 LEN 3        ; sets ROM 100, 101, 102 to instructions
*                                    ; from ROM 5, 6, 7
*RAM 10 TO 12 = .1, .3, .5, .7       ; error . . . last item will be unused
ERR 97:EXECESSIVE DATA
*RAM 10 LEN 4
RAM 10 = 0.09999996
RAM 11 = 0.30000000
RAM 12 = 0.50000000
RAM 13 = 0.41719673
*
*ROM 10 TO 11 = LDA .P030, .P031
*ROM 10 TO 11
ROM 010 = LDA  .P030,.P031,R00,NOP
ROM 011 = LDA  .P030,.P031,R00,NOP
*
```

Symbol Table Commands

The SM2920 module maintains a symbol table to allow you to refer to memory addresses and other values by using symbolic references in the SM2920 commands.

This section gives details on the following commands: Display Symbol Table, DEFINE, Change, and REMOVE symbol.

The SM2920 module symbol table receives symbols from two sources:

1. Symbol tables associated with user programs can be copied to the SM2920 symbol table when programs are loaded (if you specified the DEBUG control option when you assembled your program).

and

2. You can define additional symbols for use during the simulation session.

Corresponding to each symbol in the table is a number that you can interpret and use either as an address or as a numeric value (variable or constant). The next few paragraphs discuss the kinds of symbols that can appear in the table, and the interpretation of the corresponding symbol table quantity (address or value).

Instruction labels can be LOADed with the program code. The symbol table gives the address of the instruction carrying that label.

A program variable is a symbol for a location in data memory (RAM) that can have its content changed as a result of an instruction in the program. The symbol table gives the address where the variable value is stored.

Initially (before any code is LOADed), the symbol table is empty. Any symbols loaded or defined are stored in the order in which they were loaded or defined.

In addition to the symbols stored when the program code is LOADed, you can use the DEFINE command to define new symbols for your use during the simulation session. The rules for user-defined symbols are as follows:

a. The name of the new symbol (symbol-name) can be defined with a maximum of 31 characters. Any characters after the first 31 are accepted but ignored.

b. The first character in the new symbol-name must be an alphabetic character, or one of the characters @ or ?, or the underline character __. The remaining characters after the first can be any of these or numeric digits; or the dollar sign ($), which is printed as entered but internally ignored.

c. The new symbol-name cannot duplicate a symbol name already present.

d. When you define a new symbol, you also specify the value corresponding to it in the table. You can treat the value you assign as an address or as a numeric value for use other than addressing.

The term "expression" as used in the DEFINE symbol command is discussed in Chapter 2.

Once a symbol has been defined or loaded, any reference to that symbol is equivalent to supplying its corresponding value.

To display the value of a symbol from the symbol table, enter the appropriate symbolic reference followed by a carriage return. The Simulator displays the symbol table value on the next line.

To display the entire SM2920 symbol table, enter the command SYMBOL followed by a carriage return. Symbols are displayed in the order loaded and/or defined. The value corresponding to each symbol is also displayed.

You can change the value corresponding to an existing symbol by entering a command of the form:

symbolic-reference = expression

However, this will not change the value used in your program, e.g., if this symbol is a variable in data memory. It will only change the value in the symbol table used for display commands, for example.

To delete one or more symbols from the table, use a command of the form:

REMOVE list of symbolic-references

The symbolic-references in the list are separated by commas. The SM2920 module searches the table for each reference, deleting each symbol name.

Note that deleting a symbol from the SM2920 symbol table makes that symbol inaccessible to the Simulator, but does not affect the program code.

To delete the entire SM2920 symbol table, enter the command REMOVE SYMBOLS.

Display Symbol Table Command

```
|symbolic-reference|
|SYMBOL            |
```

where

symbolic-reference	is a period followed by an identifier. The value corresponding to that symbol is displayed.
SYMBOL	is a command keyword calling for the display of the entire SM2920 symbol table.

Examples:

```
*ABC
.ABC = 119
*SYMBOL
.ABC = 119
.DEF = 4
.X = 36
*
```

DEFINE Symbol Command

```
DEFINE symbolic-reference = expression
```

where

DEFINE	is a command keyword that tells the SM2920 module to enter the new symbol in the table and assign it in the initial value given.
symbolic-reference	is a period followed by an identifier. The symbol defined may not duplicate a symbol already in the symbol table.
=	is the assignment operator.
expression	is described earlier in Chapter 2.

Examples:

```
DEFINE .ABC = .234
DEFINE .TEMP = .ABC + .2
DEFINE .CURRENT_DAR = DAR
```

Change Symbol Command

```
symbolic-reference = expression
```

where

symbolic-reference	is a period followed by an identifier. The symbol referenced symbol table will be given a value equal to the evaluated expression.
=	is the assignment operator.
expression	is described earlier in Chapter 2.

Examples:

```
.ABC = 20H
.TEMP = .ABC + .DEF
```

REMOVE Symbol Command

```
REMOVE symbolic-reference [,symbolic-reference], . . .
REMOVE SYMBOLS
```

where

REMOVE	is a command keyword causing the symbols that follow to be deleted from the SM2920 symbol table.
symbolic-reference	is a period followed by an identifier. Several symbols can be deleted with one command by entering the symbols to be deleted as a list with the symbolic-references separated by commas.
SYMBOLS	is a keyword that (as the object of REMOVE) deletes all symbols in the symbol table.

Examples:

```
REMOVE .ABC
REMOVE .DEF,.TEMP
REMOVE SYMBOLS
```

Simulation Control Commands

Commands directly relevant to simulation fall into two general categories, simulation control and trace control. They include:

a. Specify zero as the starting address and initial time of the simulation, or that simulation is to start from the next instruction after where the last simulation stopped, either via a breakpoint or via your interrupting it with the ESCape key.

b. Specify items to be traced during simulation. (See TRACE command.)

c. Specify conditions that will cause trace records to be collected. (See QUALIFIER command.)

d. Specify and display the halting conditions for stopping simulation and returning control to the console. (See TILL clause and BREAKPOINT.)

e. Specify the input signal as a function of various waveforms, constants, and voltage levels.

f. Print trace data after simulation halts. (See Print Trace.)

g. Abort a simulation with the ESCape key.

Simulation allows you to execute your program or an experimental fragment of code, preserving the values of trace items which you have specified, up to the most recent 400 such records. This allows you to exercise your programs and learn about the details of the execution.

The commands in this section are:

Command	Purpose
SIMULATE command	Start simulation and optionally set halting conditions, or clear PC and TIME to zero and empty TRACE buffer (by saying FROM 0).
TILL clause/ONCE/FOREVER	Set halting conditions in the BREAKPOINT.
Display BREAKPOINT	Display halting conditions as currently set.
Set BREAKPOINT	Set new halting conditions.
Specify and Display Input Functions	Create or display the functions defining input signals on each input channel.

Before explaining these commands in detail, here is an example of a simple simulation with defaults. (Be sure to read the full explanations given after this example.)

To initialize for simulation, you LOAD your program code or enter it from the console into simulated ROM memory. After the code has been loaded, the SM2920 module defaults for simulation are:

a. The program counter (PC), COUNT, TQUAL, TIME, and BUFFERSIZE are set to zero.

b. The BREAKPOINT is set to NEVER. This means no halting conditions are enabled.

Now you can begin simulation by modifying any defaults and entering the command SIMULATE followed by a carriage-return. At the command SIMULATE, the following occurs:

- Simulation begins with the instruction at the address defined by PC.
- Simulation will continue until you press the ESCape key.

Now, if you press the ESCape key, the following happens:

- The SM2920 module completes the current instruction.
- The message SIMULATION TERMINATED is displayed.
- The message PROCESSING ABORTED is displayed, acknowledging the user abort (via ESC).

This is the simplest case of starting and stopping simulation. Whenever the BREAKPOINT is set to NEVER, you can enter the command SIMULATE to start simulation at the current PC address, and press the ESCape key to halt simulation.

Specifying the Starting Address

Instead of starting simulation at the address currently in the PC, you may specify a new starting address. There are two ways to do this. You can set the PC directly to any desired address with a command of the form PC = address, then enter the SIMULATE command (without FROM 0) to start simulation at that address. Or you can specify zero as the starting address as part of the SIMULATE command:

```
SIMULATE FROM 0
```

To start simulation with the instruction at location 30, you could enter

```
PC = 30
```

and then

```
SIMULATE
```

To restart at zero, you could enter

```
SIMULATE FROM 0
```

Halting Conditions in SIMULATE and Set BREAKPOINT Commands

Whether assigned in a SIMULATE, or set directly, the BREAKPOINT supplies halt condition specifications for subsequent SIMULATE commands.

You can use either the TILL clause of the SIMULATE command or the Set BREAKPOINT command to specify the conditions which will cause halting if they are met.

The simulation halting conditions control subsequent SIMULATEs until another FOREVER or ONCE or TILL clause occurs in a SIMULATE, or until a new Set BREAKPOINT command is used.

Set BREAKPOINT Command

```
BREAKPOINT = |ALWAYS                                    |
             |condition [AND/OR condition] . . .        |
             |NEVER                                     |
```

where

ALWAYS means a break is forced after every instruction simulated.

NEVER means no break occurs at all (unless you hit the ESCape key).

conditions means one or more relational expressions separated by a connector keyword, i.e., AND or OR, in the same manner as conditions are set for the qualifier. You may have any number of conditions connected by ANDs and ORs in the breakpoint or qualifier.

A relational expression has the form exp relation exp, where exp can be any legal expression and relation can be any one of the following: less than (<), less than or equal (<=), equal (=), greater than or equal (>=), not equal (<>), or greater than (>).

Examples:

```
BREAKPOINT = ALWAYS
BREAKPOINT = NEVER
BRE = PC > 70 AND TIME > 0.08
B = RAM .VAR23 < 0 OR RAM .VAR45 > 0
B = RAM .VAR23 > 0.875 OR RAM .VAR13 < 0.25
```

SIMULATE Command

```
SIMULATE [FROM 0] [|FOREVER                                  |]
                  [|ONCE                                     |]
                  [|TILL condition [AND/OR condition] . . .  |]
```

The SIMULATE command starts simulation; the TILL clause or the ESCape key can halt simulation. If a FROM 0 clause appears, COUNT, TQUAL, BUFFERSIZE, TIME, and the PC are set to zero before simulation starts. If no FROM clause is present, the most recent Set PC command or the most recent halting point will govern where simulation will resume. Startup default is zero.

The FOREVER clause sets the BREAKPOINT to NEVER. If FOREVER is specified, simulation can be halted only by using the ESCape key, which will abort processing any command not involving disk input/output. The ONCE clause sets the breakpoint to ALWAYS. If either FOREVER, ONCE, or TILL phrases appear in a SIMULATE command, they cause previous contents of the breakpoint register to be lost, and the new values to be specified thereafter.

Examples:

```
SIMULATE FROM 0 FOREVER
SIMULATE ONCE
SIMULATE FROM 0 TILL PC = 62
```

```
S TILL RAM 20 = 0.5 OR PC = 126
SIM TILL T-TQUAL > 0.008
SIMULATE TILL OVF = 1 AND PC > 100
S TILL T > 0.005
```

Display BREAKPOINT Command

You can display the breakpoint by entering the word BREAKPOINT after you get the SM2920 prompt (*).

Examples:

```
*BREAKPOINT
BREAKPOINT = RAM 20 = 0.5 OR PC = 126
*B
BREAKPOINT = T – TQUAL > 0.008
*BRE
BREAKPOINT = OVF = 1 AND PC > 100
*B
BREAKPOINT = RAM .VAR23 > 0.875 OR RAM .VAR13    0.25
```

Specify and Display Input Functions

IN | 0 | = [IF condition [AND/OR condition] . . . THEN expression ELSE] . . . expression
| 1 |
| 2 |
| 3 |

The SM2920 Simulator enables you to specify the input functions you expect to see on the four input pins. The constants PI, HPI, TPI, and ONE can be useful in specifying these functions. PI stands for 3.1415928, ONE means 0.11111111111111111111111B the largest positive number which can be stored in a 2920 RAM variable. HPI and TPI mean half-PI and twice-PI.

After you set up these functions, whenever the Simulator encounters an input iocode (e.g., IN2, IN3), indicating that channel should be sampled, it evaluates the function using the current values for the variables you used. You may use time, processor variables, or user variables. You can display the current function setting by typing the channel desired followed by carriage-return, e.g., IN3. After a simulation halts, you can display the last sample value computed during that simulation, by typing IN carriage-return.

Certain built-in functions are also available: they are SIN, COS, EXP, LOG, ABS, SQR, SAW, and SQW. EXP and LOG are natural, not decimal, i.e., EXP indicates exponentiation of e (2.71828) and LOG is to the base e. ABS is absolute value, SQR is square root of absolute value.

SAW and SQW are saw wave and square wave versions of sine, i.e., if $-HPI < X <= HPI$, then SAW(X) = X/HPI, or if $HPI < X \leq 3*HPI$ then SAW(X) = 2 – X/HPI. SAW repeats with period TPI. For SQW, if $0 < Y \leq PI$, then SQW(Y) = 1. If $PI < Y \leq TPI$, then SQW(Y) = –1. SQW repeats with period TPI.

The IF/THEN/ELSE construction allows you to specify conditions to be tested. If the conditions between the word IF and the word THEN are met, the value after the THEN is used. If the specified conditions are not met, then the value after the ELSE is used.

Examples:

```
*IN0 = SIN(4400*TPI*TIME)                    ; specifies sine wave at 4400 Hz
*IN1 = SAW(1000*TPI*TIME)*EXP(-U)            ; decreasing 1000 Hz saw wave
*IN2 = COS(2200*TPI*TIME)                    ; cosine at 2200 Hz
*IN3 = IF 0.1 < TIME AND TIME < 0.15 THEN 1 ELSE 0
```

The initial asterisk in each line is the SM2920 prompt character. The function given for IN3 above is a pulse only.

TRACE Control Commands

The SM2920 module can record the value of any register and status bits in a trace data buffer during simulation. The commands in this section allow you to specify what items or expressions are to be traced, to control the display of trace data, and to enable or disable trace data collection during simulation by setting conditions which qualify or inhibit such collection.

The commands in this section are as follows:

Command	Purpose
CONSOLE	Display trace information during a simulation.
Set TRACE Items	Cause trace data to be collected on specific variables, registers, flags, etc.
MOVE, OLDEST, NEWEST	Move trace buffer pointer to record to be displayed.
PRINT	Display one or more records from the trace buffer.
Set Qualifier Register	Specify match conditions which determine when trace is to be recorded.
Display TRACE Items	Display current trace items.
Display Qualifier	Display current conditions in QUALIFIER. [The default (initial and reset) value of QUALIFIER is ALWAYS, meaning collect trace after each instruction simulated.]
GRAPHICS	Cause trace data to be displayed as curves in a graph (GRAPHICS ON) or as values in a table (GRAPHICS OFF).

The buffer contains as many records of trace data as possible up to 400, which is the upper limit on the product of number-of-trace-items and the number-of-records-kept. As mentioned in Chapter 1, if four trace items are specified, up to 100 records can be kept in the trace buffer. If more are collected, the most recent 100 are kept. If eight trace items are specified, only 50 records can be kept.

Each record in the buffer contains the value of whatever expressions were named as trace items in the Set TRACE command.

The trace buffer pointer controls the display of data from the buffer. When simulation is started after a SIMULATE FROM 0 command, the buffer is cleared of data, and the buffer pointer is reset to the oldest end (just before the earliest entry). The buffer is initially empty. After the simulation halts, the buffer pointer is at the newest end of the buffer, just past the most recently collected record. If FROM 0 is not used, the buffer pointer's position is not reset.

CONSOLE Command

The CONSOLE command controls the display of trace information during a simulation. When the command CONSOLE ON is in effect, such data are displayed. When the command CONSOLE OFF is in effect, such data are not displayed, and the

simulation runs faster. The LIST command can be used to capture data for later review and analysis, regardless of which console setting is operative. The PRINT command is similarly independent of the CONSOLE command, because the trace data are collected into the trace buffer anyway. Thus you can direct copies of trace data records to the console, a file, or both, using combinations of these commands.

Set and Display TRACE Items Command

This command controls the items whose values are to be collected into the trace buffer during simulation. An entry can be any legal expression, e.g., the contents of a RAM location or a processor reference. To set the trace items, use the following command:

TRACE = item1, item2, item3, . . .

where each item is an expression as discussed in Chapter 2.

Example:

TRACE = PC, TIME, IN3/OUT3, RAM .VAR24

To display the current items set for TRACing during the next simulation, simply enter TRACE and a carriage-return. The names of those items will be displayed on the next line.

To display data from the buffer of records collected in the last simulation, MOVE the pointer to the desired entry and enter a PRINT command.

The number of trace items determines the number of columns during trace data display. How many can fit on the screen depends on the BASE at the time of display.

Moving the Buffer Pointer

The pointer movement commands are MOVE, OLDEST, and NEWEST. The OLDEST command (followed by carriage-return) moves the pointer to the oldest record in the buffer. The NEWEST command moves the pointer to the record most recently collected.

The MOVE command has the following form:

MOVE [[+/−] primary]

Primary was defined under expression in an earlier section, and is just a restricted form of expression. The value entered is used as the number of records between the current pointer position and the desired position. Movement in a plus (+) direction is toward the newer end of the buffer; if neither (+) nor (−) is entered, a newer movement is assumed as the default. Movement in a minus direction is toward the older end of the buffer.

For example, if the pointer is pointing at record 100 and you issue the command MOVE 10, the pointer is moved to point to record 110. Under the same initial conditions, if you issue the command MOVE −10, the pointer is moved to point to record 90. If the number you enter is larger than the number of records between the current pointer location and the oldest (for −) or newest (for +), the pointer is moved only to the newest or oldest end, respectively. In short, you cannot move the pointer outside the range of buffer locations. If the MOVE command has no expression following it, MOVE 1 is executed.

Displaying TRACE Data

The PRINT command displays one or more records from the buffer. This command has the form:

PRINT [[+/−] primary | ALL]

Primary was defined under expressions, and is a restricted expression.

With (+) or no sign, records newer than the current pointer position are displayed and the pointer is moved. With (−), records older than the current pointer position are displayed. The command PRINT without an expression modifier is equivalent to PRINT 1 (one record is displayed) and the pointer advances one.

The PRINT command displays the number of records requested, then (for positive values only) moves the pointer to point to the next entry just past the last one displayed. As an illustration, the commands

```
OLDEST
PRINT 10
PRINT 10
```

are equivalent to the commands

```
OLDEST
PRINT 20
```

The pointer now points to the 21st record.

The command PRINT ALL displays the entire trace buffer.

The trace data are displayed in columns separated by spaces. The SM2920 module displays a header each time PRINT is used, giving the name of each display above the column for that display. The trace display header depends on the trace items specified. The following illustration shows the type of display for trace items. (Other commands shown are explained later in this chapter.)

```
*simulate till pc = 0 and t > .4
   TIME
SIMULATION BEGUN                OUT1
     0.00000050                  0.06250000
     0.00000100                  0.81250000
     0.00000150                 -0.92187500
     0.00000200                  0.15234375
     0.00000250                  0.75781250
     0.00000300                 -0.95312500
     0.00000350                  0.23437500
     0.00000400                  0.69921875
     0.00000450                 -0.97656250
     0.00000500                  0.32031250
     0.00000550                  0.63671875
PROCESSING ABORTED
*: pressed the ESCape key
*print 4
   TIME                         OUT1
*print -4
   TIME                         OUT1
     0.00000400                  0.69921875
     0.00000450                 -0.97656250
     0.00000500                  0.32031250
     0.00000550                  0.63671875
*oldest
*print 4
   TIME
     0.00000050                  0.06250000
     0.00000100                  0.81250000
```

```
    0.00000150                    -0.92187500
    0.00000200                     0.15234375
*
*
*
*break?
BREAKPOINT = PC=0 AND T > .4
*qua
QUALIFIER = ALWAYS
*
```

NOTE

Whenever SIMULATE FROM 0 is used, the PC, COUNT, and TQUAL are set to zero and the trace buffer is emptied (BUFFERSIZE = 0), and the simulation timer is reset (TIME = 0). Thus, the trace buffer and simulation time record the history since simulation was started at zero.

MOVE, OLDEST, and NEWEST Commands

MOVE [[+/−] primary]

OLDEST

NEWEST

where

MOVE	is a command keyword that moves the buffer pointer one or more records forward (toward the most recent records) or backward (toward the earliest records).
+	is a unary operator specifying a movement toward the newest end of the buffer. Plus is the default.
−	is a unary operator specifying movement toward the oldest end of the buffer.
primary	is a restricted expression that gives the number of records to be included in the MOVE.
OLDEST	is a command keyword that moves the pointer to the earliest entry in the buffer.
NEWEST	is a command keyword that moves the pointer to just past the most recent entry in the buffer.

Examples:

MOVE	The buffer pointer moves one record toward NEWEST.
MOVE +6	The buffer pointer moves six records toward NEWEST.
MOVE −11	The buffer pointer moves eleven records toward OLDEST.
OLDEST	The buffer pointer is repositioned at OLDEST.
NEWEST	The buffer pointer is repositioned at NEWEST.

PRINT Command

PRINT ALL

PRINT [[+/−] primary]

where

PRINT	is a command keyword calling for a display of one or more records from the trace data buffer. The records are values for the trace items you typed in when setting TRACE.
ALL	is a function keyword indicating that the entire trace buffer contents are to be displayed.
+	is a unary operator directing the display of records newer than the current buffer pointer location. Plus is the default. The pointer is repositioned past the last record printed.
–	is a unary operator directing the display of records older than the current buffer pointer location. The pointer is not moved.
primary	is a restricted expression giving the number of entries (records) to be displayed.

Examples:

```
PRINT
PRINT ALL
PRINT +5
PRINT 5
PRINT -10
```

Set and Display Qualifier Command

```
QUALIFIER = | ALWAYS
            | condition [AND/OR condition] . . .
            | NEVER
```

where

ALWAYS	means trace is collected after every instruction simulated.
NEVER	means trace is not collected at all.
conditions	means one or more relational expressions separated by a connector keyword, i.e., AND or OR, in the same manner as conditions are set for the breakpoint. You may have any number of conditions connected by ANDs and ORs in the breakpoint or qualifier. Trace is collected for all trace items while the conditions in the qualifier are met. A relational expression has the form exp relation exp, where exp can be any legal expression and relation can be any one of the following: less than (<), less than or equal (<=), equal (=), greater than or equal (>=), not equal (<>), or greater than (>). To display the current setting of the qualifier, you type QUAL and carriage-return.

Examples:

```
QUALIFIER = ALWAYS
QUALIFIER = NEVER
QUA = PC > 70 AND TIME > 0.08
Q = RAM .AR23 < 0 OR RAM .VAR45 > 0
```

GRAPHICS Command

When GRAPHICS OFF is in effect, the display of trace information is tabular as described above. When GRAPHICS ON is in effect, each trace item's changing value is displayed as a curve, running vertically down from the top of the screen.

The graph of zeros corresponds to first trace item, one's corresponds to second trace item, and so forth.

This format permits continuous graphic display on a line-printer as well as the console.

XSIZE controls the trace display width, both tables and graphs. Its minimum value is 4 and its maximum, 253. The default is 79.

For the horizontal scale, the leftmost value is −1 and the rightmost, +1. This range corresponds to the output limits of the 2920 chip. The range is partitioned equally by the value of XSIZE-2.

In general, you can always transform your range of interest into [−1, +1] by some suitable offset and scaling. For example, if

$$0.8 \leqslant X \leqslant 0.9$$

then specify $(X - 0.85)/0.05$ as your trace item.

The vertical axis will normally be labeled with the trace record number. However, if TIME was the first trace-item specified, then this axis will be labeled with time in seconds, and there will be no curve drawn with "0"s.

CHAPTER 4
A COMPLETE EXAMPLE SIMULATION SESSION

The example given in this chapter shows many aspects of the 2920 Simulator's capabilities, using many of the command forms shown in previous chapters. The sections just below provide a few insights for making test simulations run faster.

Simulation Speed

The speed of simulation varies with the complexity of the input signals, the breakpoints, and trace. Exclusive of I/O time requirements, 2920 instructions will typically be simulated at a rate of several hundred per second. If trace data is output during simulation, this rate may be significantly slower.

Time constants for sample-and-hold input and output capacitors are assumed to be zero. This means a single input or output instruction is sufficient to sample or output a signal, which would not be the case in actual use. However, the *internal* processing of the 2920 is simulated exactly.

To effect the simulation of each single instruction, the SM2920 module performs the following steps in order:

- If an input iocode is involved in the 2920 instruction about to be executed, the appropriate input signal function is evaluated using the current state of all system variables. The instruction pointed to by the PC (Program instruction Counter) is then executed.

- If an EOP (End Of Program code) is about to take place, PC is set to zero and TIME is set to [(time at last EOP)+TPROG (the time for one pass through program)]; otherwise, if an EOP is not happening, PC is incremented by 1 and TIME is incremented by the time per instruction, called TINST, which is TPROG divided by SIZE (the number of instructions in the program).

- The conditions in the QUALIFIER are evaluated to see whether to collect trace, and if so, trace is collected and possibly displayed (if CONSOLE is ON).

- The BREAKPOINT is evaluated to see whether to stop simulation.

The Simulator acts as if it were simulating a program of length SIZE with an instruction time of TPROG/SIZE. An EOP may be encountered before that many instructions have been simulated. If so, the instruction containing the EOP and the next three instructions are executed, and the remaining instructions are simulated as if they were NOPs (no-operation codes). This means that the TIME skips ahead to what the time at the next program pass should be.

The SM2920 module can be run under the ISIS SUBMIT facility, permitting unattended tests of one or more 2920 programs. See the ISIS manual listed in the Preface for a more detailed discussion of this facility.

Increasing Simulation Speed

Simulation can be made to run faster by minimizing the amount of computation the Simulator must perform between each instruction. For example, consider:

```
*IN0 = IF T < 0.001 THEN SIN(60*TPI*T) ELSE SIN (70*TPI*T)
*SIMULATE FROM 0 TIL T < 0.002
```

This specification of input-channel-0 describes a 60 Hz sine wave for the first millisecond, and a 70 HZ sine wave thereafter. The Simulator must repeatedly compare the time with 0.001. A faster version of the same experiment could be:

```
*IN0 = SIN (60*TPI*T)                ; only when t   0.001, however
*SIMULATE FROM 0 TIL T > 0.001
*IN0 = SIN(70*TPI*T)                 ; from now on
*SIMULATE TIL T > 0.002              ; finish experiment
```

Note that the Simulator does premultiply constants on the left (only!) side of a variable appearing in a product. This means

```
*IN0 = SIN(60*2*PI*T)
```

takes the same time to compute during simulation as

```
*IN0 = SIN(376.9911*T)
```

Another place where computation time is important is in evaluating breakpoint and qualifier conditions. For example:

```
*SIMULATE TIL (OUT0 – OUT1) > 0.5
```

The Simulator will check the value of (OUT0 – OUT1) between each instruction. Since OUT0 and OUT1 only change once per program pass, a faster version would be

```
*SIMULATE TIL PC = 0 AND (OUT0 – OUT1) > 0.5
```

The Simulator will only compute (OUT0 – OUT1) once per program pass, when pc = 0.

Similarly, the command

```
*SIMULATE TILL PC = 0 AND T > 0.001
```

is faster than the command

```
*SIMULATE TILL T > 0.001
```

because the floating point check on T is slower than the integer check on PC. Faster still is

```
*SIMULATE FOREVER
```

This simulation will stop only when you press the ESCape key.

Simulation speed can be increased for testing purposes by replacing a series of input iocodes in the program with a single input iocode, with the rest changed to NOPs. The Simulator assumes a time constant of zero for the sample-and-hold, making a single input iocode sufficient for simulation. In actual use, the program must be restored to having the full number of input iocodes necessary.

To obtain dumps after fixed time intervals, you can use the system variable which gives the time the last trace datum was recorded, called TQUAL. For example,

```
*QUALIFIER = PC=0 AND T – TQUAL > 0.001
```

will provide trace data sampling roughly once per millisecond. The following will also work but is slower:

```
*QUALIFIER = PC=0 AND (T MOD 0.001) < TPROG
```

The pages following give a complete sample session of using the Simulator.

```
ISIS-II 2920 SIMULATOR, V1.1
*evaluate pi   ; what is value of pi?
 3.1415926E+0     3.1415926E+0     3.1415926E+0
*; numbers not between +1 and -1 are displayed in scientific notation
*eva sin(pi/(7-1))/10   ; value should be about 1/20
 0 00001100110011001100110118     0.05000000D     0.0CCCCDH
*; numbers between +1 and -1 are displayed as above
*
*dar   ; can display 2920 registers by naming them
DAR =  0.12500000
*; display is in default display base - initially decimal
*base = b   ; change default display base to binary
*dar   ; look at dar in binary
DAR =  0.00100000000000000000000
*base = h   ; change default display base to hexadecimal
*dar   ; look at dar in hexadecimal
DAR =  0.200000
*base = d   ; change back to decimal
*
*; we can also change the default input suffix
*suffix   ; display initial suffix
SUFFIX = D
*suffix = b   ; set default suffix to binary
*dar = .11   ; i.e. 0.11b
*evaluate dar   ; should be 0.11b
 0 11000000000000000000000008     0.75000000D     0.C00000H
*suffix = d   ; we prefer decimal default however
*
*load :f1:band2.hex   ; assembler output of my filter program
LOCATION 192
WARN C2:NO EOP
*symbols   ; compare with assembler listing
.P011= 0.00000000
.P012= 1.00000000
.P010= 2.0000000E+0
.P031= 3.0000000E+0
.P032= 4.0000000E+0
.P030= 5.0000000E+0
.TEMP= 6.0000000E+0
.P051= 7.0000000E+0
.P052= 8.0000000E+0
.P050= 9.0000000E+0
*; note that user defined symbols are always preceded with a period
*; to avoid confusion with command keywords.
*rom 0 to 36   ; compare with assembler listing
ROM 000 = SUB DAR,DAR,R00,NOP
ROM 001 = LDA .P011,.P012,R00,NOP
ROM 002 = LDA .P012,.P010,R00,NOP
ROM 003 = LDA .P010,.P011,R06,NOP
ROM 004 = SUB .P010,.P011,R00,NOP
ROM 005 = SUB .P010,.P010,R07,NOP
ROM 006 = SUB .P010,.P012,R00,IN1
ROM 007 = SUB .P010,.P012,R03,NOP
ROM 008 = ADD .P010,.P012,R05,CYTS
ROM 009 = ADD .P010,.P012,R07,NOP
ROM 010 = LDA .P031,.P032,R00,CYT7
ROM 011 = LDA .P032,.P030,R00,NOP
ROM 012 = SUB .P030,.P030,L02,CYT6
ROM 013 = SUB .P030,.P030,R03,NOP
ROM 014 = SUB .P030,.P030,R01,CYT5
ROM 015 = SUB .P030,.P030,R03,NOP
ROM 016 = SUB .P030,.P031,R00,CYT4
ROM 017 = ADD .P030,.P031,R06,NOP
ROM 018 = ADD .P030,.P031,R08,CYT3
ROM 019 = LDA .TEMP,.TEMP,R00,NOP
ROM 020 = LDA .P051,.P052,R00,CYT2
ROM 021 = LDA .P052,.P050,R00,NOP
ROM 022 = LDA .P050,.P051,R04,CYT1
ROM 023 = SUB .P050,.P051,R00,NOP
ROM 024 = ADD .P050,.P050,R06,CYT0
```

```
ROM 025 = SUB .P050,.P052,R00,NOP
ROM 026 = SUB .P050,.P052,R03,OUT1
ROM 027 = ADD .P050,.P052,R06,NOP
ROM 028 = ADD .P050,.P052,R07,NOP
ROM 029 = ADD .P010,DAR,R08,NOP
ROM 030 = ADD .P030,.P010,R04,NOP
ROM 031 = ADD .P050,.P030,R03,NOP
ROM 032 = LDA DAR,.P050,R00,NOP
ROM 033 = LDA .P011,.P011,R00,NOP
ROM 034 = LDA .TEMP,.TEMP,R00,OUT0
ROM 035 = LDA .P011,.P011,R00,NOP
ROM 036 = XOR .P011,.P011,R01,IN0
*; This program takes input from in1, converts it, outputs it to out1
*; for debug, bandpasses it, and outputs it to out0.
*tprog = 1/13020  ; final version of program will run at 13020 HZ
*size  ; gives number of instructions in program
SIZE =  1.9200000E+2
*tinst  ; should be tprog/size - time per instruction
TINST =  0.00000040
*rom 32 = lda dar,.p050,eop  ; speed up simulation by not simulating
*; instructions after first 36.  As long as tprog and size are what
*; they will be in the final program, simulation will be accurate.
*rom 30 length 4  ; verify above rom change
ROM 030 = ADD .P030,.P010,R04,NOP
ROM 031 = ADD .P050,.P030,R03,NOP
ROM 032 = LDA DAR,.P050,R00,EOP
ROM 033 = LDA .P011,.P011,R00,NOP
*
*ram 0 to 40 = 0  ; initialize ram to known state for debug
*dar  ; examine initial value of dar
DAR =  0.00000000
*evaluate dar  ; examine initial value of dar in all bases
 0.00000000000000000000000000B    0.000000000D    0.00000000H
*
*in1  ; examine initial specification for input 1
IN1 = 0 =  0.00000000
*in1 = sin(4400*tpi*time)  ; specify sine wave at 4400 HZ for input 1
*in1  ; examine input 1 specification and its current value
IN1 = SIN(4400*TPI*TIME) =  0.00000000
*
*trace  ; examine default trace specification
TRACE = TIME
*trace = time, out0, out1  ; items to be traced during simulation
*qualifier  ; examine default qualifier
QUALIFIER = ALWAYS
*qualifier = pc=0  ; when we want trace collected - once per pass
*breakpoint  ; examine default breakpoint
BREAKPOINT = NEVER
*breakpoint = pc=0 and t>20/13020  ; we want simulation to stop after
*; 20 passes through program.  We included the pc=0 check because
*; it's faster than the t>20/13020 check.
*
*simulate  ; let's do it!
   TIME              OUT0              OUT1
SIMULATION BEGUN
     0.00007680        0.00195313        0.06445313
     0.00015361        0.00195313        0.81445313
     0.00023041       -0.00195313       -0.92382813
     0.00030722        0.00195313        0.15039063
     0.00038402       -0.00195313        0.75976563
     0.00046083       -0.00195313       -0.95507813
     0.00053763        0.00195313        0.23632813
     0.00061444       -0.00195313        0.70117188
     0.00069124       -0.00195313       -0.97851563
     0.00076805        0.00195313        0.31835938
     0.00084485       -0.00195313        0.63476563
     0.00092166        0.00195313       -0.99414063
     0.00099846        0.00195313        0.40039063
     0.00107527       -0.00195313        0.56445313
     0.00115207        0.00195313       -0.99804688
     0.00122888        0.00195313        0.47851563
     0.00130568       -0.00585938        0.49414063
     0.00138249        0.00585938       -0.99804688
     0.00145929        0.00195313        0.55273438
     0.00153610       -0.00976563        0.41601563
     0.00161290        0.00976563       -0.99414063
SIMULATION TERMINATED
*
*print -3  ; the trace data above is also saved in the trace buffer
```

```
   TIME              OUT0              OUT1
    0.00145929        0.00195313        0.55273438
    0.00153610       -0.00976563        0.41601563
    0.00161290        0.00976563       -0.99414063
*oldest  ; let's look at the oldest records in the trace buffer
*print 3  ; oldest three records
   TIME              OUT0              OUT1
    0.00007680        0.00195313        0.06445313
    0.00015361        0.00195313        0.81445313
    0.00023041       -0.00195313       -0.92382813
*p  ; next record
   TIME              OUT0              OUT1
    0.00030722        0.00195313        0.15039063
*move 5  ; skip over next five records
*p 2  ; display next two records
   TIME              OUT0              OUT1
    0.00076805        0.00195313        0.31835938
    0.00084485       -0.00195313        0.63476563
*print all  ; this displays all the trace data in the trace buffer
   TIME              OUT0              OUT1
    0.00007680        0.00195313        0.06445313
    0.00015361        0.00195313        0.81445313
    0.00023041       -0.00195313       -0.92382813
    0.00030722        0.00195313        0.15039063
    0.00038402       -0.00195313        0.75976563
    0.00046083       -0.00195313       -0.95507813
    0.00053763        0.00195313        0.23632813
PROCESSING ABORTED
*; We aborted the above display with the escape key.  Control S and
*; control Q can also be used to stop and start display.
*buffersize  ; this system variable gives the number of records in
BUFFERSIZE =  2.1000000E+1
*; the trace buffer
*
*; the default display base also affects trace display ...
*base = h  ; hexadecimal
*newest  ; position trace display pointer just past newest trace record
*p-5  ; look at 5 most recent trace data records in hexadecimal
   TIME              OUT0              OUT1
    0.005592         -0.018000          0.7E8000
    0.005A9A          0.018000         -0.FF8000
    0.005FA3          0.008000          0.8D8000
    0.0064AB         -0.028000          0.6A8000
    0.006984          0.028000         -0.FE8000
*base = d  ; back to good old decimal
*
*; Trace data can be displayed in tabular format as above,
*; or in graphic format.
*graph on  ; select graphic format for trace display
*p all  ; re-examine trace data
   TIME              OUT0              OUT1
->                                      1 2
0                                       1                                    2
     2                                  1
0                                       1      2
0                                       1                                2
0   2                                   1
0                                       1         2
7                                       1                             2
6  2                                    1
8                                       1             2
0                                       1                       2
  2                                     1
                                        1                 2
                                        1                       2
->2                                     1
0                                       1                   2
.                                       1                    2
0 2                                     1
0                                       1                      2
1                                       1                  2
1 2                                     1
*; We see that the time trace data is used to label the vertical axis
*; and the digits "1" and "2" are used to label the next two trace items.
*; The horizontal scale is always -1 to +1 and so our out0 signal is too
*; small to register.  Let's rescale it next time we collect trace.
*tra = time, 10*out0, out1  ; trace scaled up output signal
*simulate til pc=0 and t>=40/13020  ; simulate 20 more program passes
```

```
   TIME              10*OUT0           OUT1
SIMULATION BEGUN
->                                        1                    2
0                                    1              2
   2                                          1
0                                       1                          2
0                                    1                 2
1    2                                        1
6                                       1                          2
8                                    1              2
9     2                                             1
7                                    1                                   2
1                                    1         2
       2                                                 1
                                        1                                    2
                                          `1   2
->       2                                                    1
0                              1                                            2
                                         21
0         2                                                      1
0                           1                                               2
2                                      2         1
SIMULATION TERMINATED
*
*console off   ; the console switch controls whether trace data is
*; displayed on the console during simulation
*simulate til pc=0 and t>=60/13020   ; simulate 20 more program passes
   TIME              10*OUT0           OUT1
SIMULATION BEGUN
SIMULATION TERMINATED
*; note that the trace data was not displayed during simulation
*buffersize   ; however the data was accumulated in the trace buffer
BUFFERSIZE =  3.9000000E+1
*p-10   ; look at last 10 records accumulated during that simulation
   TIME              10*OUT0           OUT1
->                   2                                 1
0          1                                                              2
                       2                                          1
0                        2                        1
0        1                                                              2
3                  2                                                 1
9                        2             1
1          1                                                            2
7                2                                                       1
0                             21
*
*; the list command directs a copy of all console i/o to a log file
*; this includes trace data during simulation even if console is off!
*; the combination of a list file and console off is the fastest way
*; to accumulate data during simulation for further analysis
*; elsewhere with a FORTRAN program for example.
*list :f1:log.527   ; todays log file
*; today is 5/27/80 - this is the first line of :f1:log.527
*in1 = saw(1000*tpi*t)*exp(-10*t)   ; decreasing 1000 HZ saw wave
*simulate from 0 til pc=0 and t>0.010   ; simulate for 10 milliseconds
   TIME              10*OUT0           OUT1
SIMULATION BEGUN
SIMULATION TERMINATED
*; That took quite a while, all the trace data is now in the log file
*list :co:   ; this closes that log file and restores to no list file
*
*; let's see what state ram is in now
*ram 0 to 5   ; first six ram locations
RAM 00 =  0.01515383
RAM 01 = -0.01788515
RAM 02 =  0.00443846
RAM 03 = -0.00816369
RAM 04 =  0.00823420
RAM 05 = -0.00117487
*ram 0 length 6   ; first six ram locations
RAM 00 =  0.01515383
RAM 01 = -0.01788515
RAM 02 =  0.00443846
RAM 03 = -0.00816369
RAM 04 =  0.00823420
RAM 05 = -0.00117487
*ram .p011   ; named ram variable
RAM 00 =  0.01515383
*
*console on   ; back to normal
*; let's write a small sine wave generator program
```

```
*rom 0 to 191 = nop  ; fill rom with nops
*rom 0 = lda dar,0,eop
*rom 1 = lda 2,0
*rom 2 = add 0,1,r7,out0
*rom 3 = sub 1,2,r7
*rom 0 to 3  ; here's our sine wave generator
ROM 000 = LDA DAR,.PO11,R00,EOP
ROM 001 = LDA .PO10,.PO11,R00,NOP
ROM 002 = ADD .PO11,.PO12,R07,OUT0
ROM 003 = SUB .PO12,.PO10,R07,NOP
*size = 4  ; this time we really mean eop
*tprog = 4/1000000  ; assume one microsecond instruction time
*tinst  ; should be one microsecond
TINST =  0.00000100
*
*ram 0=one,1/2  ; initialize ram 0 and ram 1
*ram 0 to 1
RAM 00 =  0.99999990
RAM 01 =  0.50000000
*
*; let's monitor the behavior closely by specifying detailed trace
*graph off  ; select tabular trace display format
*trace = time, ram 0, ram 1, sqr((ram 0)*(ram 0)+(ram 1)*(ram 1))
*qualifier = always  ; let's always collect and display trace
*
*simulate from 0 once  ; take one step
   TIME             RAM 0            RAM 1            SQR((RAM 0)*(R
                                                      AM 0)+(RAM 1)*
                                                      (RAM 1))

SIMULATION BEGUN
    0.00000100       0.99999990       0.50000000       1.1180340E+0
SIMULATION TERMINATED
*s  ; another
   TIME             RAM 0            RAM 1            SQR((RAM 0)*(R
                                                      AM 0)+(RAM 1)*
                                                      (RAM 1))

SIMULATION BEGUN
    0.00000200       0.99999990       0.50000000       1.1180340E+0
SIMULATION TERMINATED
*s  ; another
   TIME             RAM 0            RAM 1            SQR((RAM 0)*(R
                                                      AM 0)+(RAM 1)*
                                                      (RAM 1))

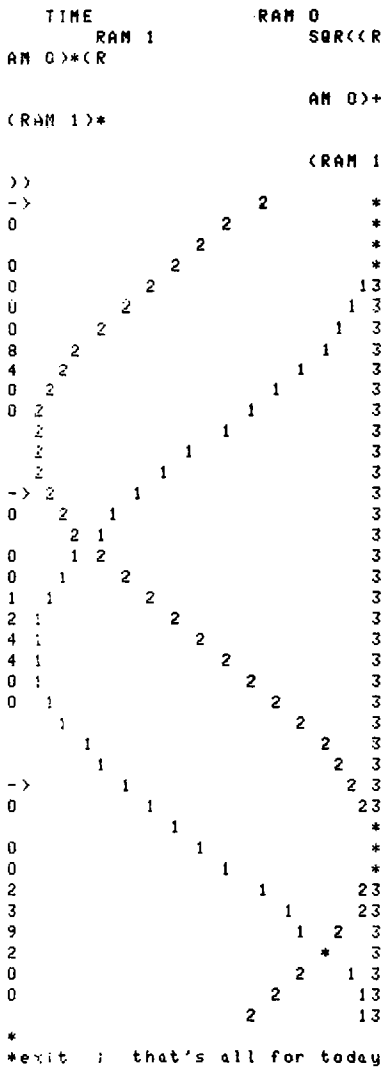
SIMULATION BEGUN
    0.00000300       0.99999990       0.50000000       1.1180340E+0
SIMULATION TERMINATED
*s til count=20  ; take 20 more steps
   TIME             RAM 0            RAM 1            SQR((RAM 0)*(R
                                                      AM 0)+(RAM 1)*
                                                      (RAM 1))

SIMULATION BEGUN
    0.00000400       0.99999990       0.49218755       1.1145620E+0
    0.00000500       0.99999990       0.49218755       1.1145620E+0
    0.00000600       0.99999990       0.49218755       1.1145620E+0
    0.00000700       0.99999990       0.49218755       1.1145620E+0
    0.00000800       0.99999990       0.48437510       1.1111342E+0
    0.00000900       0.99999990       0.48437510       1.1111342E+0
    0.00001000       0.99999990       0.48437510       1.1111342E+0
    0.00001100       0.99999990       0.48437510       1.1111342E+0
    0.00001200       0.99999990       0.47656270       1.1077508E+0
    0.00001300       0.99999990       0.47656270       1.1077508E+0
    0.00001400       0.99999990       0.47656270       1.1077508E+0
    0.00001500       0.99999990       0.47656270       1.1077508E+0
    0.00001600       0.99999990       0.46875024       1.1044122E+0
    0.00001700       0.99999990       0.46875024       1.1044122E+0
    0.00001800       0.99999990       0.46875024       1.1044122E+0
    0.00001900       0.99999990       0.46875024       1.1044122E+0
    0.00002000       0.99999990       0.46093779       1.1011192E+0
    0.00002100       0.99999990       0.46093779       1.1011192E+0
    0.00002200       0.99999990       0.46093779       1.1011192E+0
    0.00002300       0.99999990       0.46093779       1.1011192E+0
SIMULATION TERMINATED
*
*qualifier = pc=0 and t-tqual>20*tprog  ; dump each 20th program pass
*s f 0 til buffersize=20  ; simulate for twenty dumps
```

```
   TIME              RAM 0             RAM 1             SQR((RAM 0)*(R
                                                         AM 0)+(RAM 1)*
                                                         (RAM 1))

SIMULATION BEGUN
    0.00008400        0.99999990        0.29687654        1.0431374E+0
    0.00016800        0.99999990        0.13281530        1.0087814E+0
    0.00025200        0.99963369       -0.03124404        1.0001220E+0
    0.00033600        0.98173652       -0.19421273        1.0007624E+0
    0.00042000        0.93743447       -0.35217124        1.0014030E+0
    0.00050000        0.87174414       -0.49405781        1.0020136E+0
    0.00058400        0.77984614       -0.63020386        1.0026550E+0
    0.00066800        0.66687388       -0.74958901        1.0032968E+0
    0.00075200        0.53583608       -0.84898369        1.0039392E+0
    0.00083600        0.39023108       -0.92569150        1.0045820E+0
    0.00092000        0.23395259       -0.97762158        1.0052252E+0
    0.00100000        0.07905102       -1.00000000        1.0031198E+0
    0.00108400       -0.08493549       -0.99671631        1.0003286E+0
    0.00116400       -0.23914802       -0.97194990        1.0009388E+0
    0.00124400       -0.38771892       -0.92345840        1.0015496E+0
    0.00132800       -0.53367847       -0.84827754        1.0021914E+0
    0.00140800       -0.65958232       -0.75535752        1.0028030E+0
    0.00148800       -0.76956016       -0.64390933        1.0034152E+0
    0.00157200       -0.86494746       -0.50990127        1.0040586E+0
    0.00165200       -0.93432822       -0.36931843        1.0046718E+0
SIMULATION TERMINATED
*
*; It is easier to believe the signal generated is sinusoidal when
*; it can be seen graphically.
*gra on  ; select graphic format trace display
*s til buf=40  ; simulate for 20 more dumps
   TIME              RAM 0             RAM 1             SQR((RAM 0)*(R
                                                         AM 0)+(RAM 1)*
                                                         (RAM 1))

SIMULATION BEGUN
-> 1                                 2                                 3
0  1                                 2                                 3
   1                                       2                           3
0   1                                          2                       3
0     1                                              2                 3
1        1                                              2              3
7           1                                              2           3
3                 1                                              2     3
2                       1                                          2   3
0                             1                                      2 3
0                                   1                                 *
                                          1                           *
                                                1                     2 3
->                                                    1                 3
0                                                        1        2     3
                                                                 *      3
0                                                        2        1     3
0                                                            2        1 3
2                                                               2      13
SIMULATION TERMINATED
*
*; The screen width during trace display can be changed.  It is sometimes
*; easier to recognize a pattern with smaller screen widths.
*xsize  ; what is the default?
XSIZE =  7.9000000E+1
*xsize = 30     ; compress screen
*p all  ; re-examine trace
```


TIME
RAM 0
RAM 1
SQR((R
AM 0)*(R
AM 0)+
(RAM 1)*
(RAM 1
))
*exit ; that's all for today

APPENDIX A
THE HELP MESSAGES

```
*help
***  Help is available for the following items.  Type HELP followed  ***
***  by the item name.  Do not type the angle brackets.  (For more   ***
***  information on the help command, type HELP HELP.)               ***
Simulation:
   BREAKPOINT,SIMULATE
Trace:
   Collection:  QUALIFIER,TRACE
   Display:  CONSOLE,GRAPH,MOVE,NEWEST,OLDEST,PRINT
Change/Display/Define/Remove:
   <CHANGE>,<CPU$REF>,DEFINE,<DISPLAY>,<INPUT>,RAM,REMOVE,ROM,<SM$REF>
State/Mode:
   BASE,LIST,SUFFIX,
Miscellaneous:
   EVALUATE,EXIT,HELP,LOAD,SAVE,SYMBOL,
   <ADDRESS>,<ADDRESS$PARTITION>,<CONDITION>,<CONSTANT>,<EXPR>,
   <FUNCTION>,<IDENTIFIER>,<INSTRUCTION>,<NUMERIC$CONSTANT>,
   <PATHNAME>,<PRIMARY>,<SYMBOLIC$REF>
*
*
*
*
*
*help *
<ADDRESS> - A numeric value expressed as an algebraic sequence of
operand(s) and operator(s).  It typically represents a RAM or ROM
address, but in some commands it may represent a length or count.
It has the following form:
          <OPERAND> [<OPERATOR> <OPERAND>]*
<OPERAND>:
   [<UNARY$OP>] <PRIMARY>
<UNARY$OP>:
   +     -
<PRIMARY>:
   <NUMERIC$CONSTANT>      (Ex:  1979, 0.1011B, 0.FFFCH, 3.14159D)
   <SYMBOLIC$REF>          (Ex:  .X1, .STAGE$2)
   <CPU$REF>               (Ex:  PC, DAR, CY)
   <SM$REF>                (Ex:  COUNT,BUFFERSIZE,TIME)
   <INPUT>                 (Ex:  IN0, IN3)
   <CONSTANT>              (Ex:  PI, HPI, TPI, ONE)
   <FUNCTION> (<EXPR>)     (Ex:  SIN(T), SAW(1000*TPI*TIME))
   (<EXPR>)                (Ex:  (RAM .X1), (.FRED+14D))
<OPERATOR>:
   MOD     *     /     +     -     MASK

<ADDRESS$PARTITION> - A single address or a range of addresses:
(1)  <ADDRESS>
        Ex:   .START+4
(2)  <ADDRESS> TO <ADDRESS>
        Ex:   .TEMP$0 TO .TEMP$9
(3)  <ADDRESS> LENGTH <ADDRESS>
     (The second <ADDRESS> is the number of locations in the range.)
        Ex:  0 LENGTH 64

BASE - Refers to the default base in which numeric values are displayed
on the console.  Initially decimal.  See SUFFIX for default input base.
   (1)  BASE
        (display default base)
   (2)  BASE = <BASE$TYPE>
        (change default base)
<BASE$TYPE>:
   B       (Binary)
   D       (Decimal)
   H       (Hexadecimal)

BREAKPOINT - Secifies the condition for halting simulation.
The BREAKPOINT can also be set implicitly with a SIMULATE command
which includes a TIL clause.
(1)  BREAKPOINT
     (display BREAKPOINT)
(2)  BREAKPOINT = <CONDITION>
     (specify BREAKPOINT)
        Ex:  BREAKPOINT = OVF=1 OR DAR MASK 0.001B <> 0
See SIMULATE.
```

```
<CHANGE> commands:
(1)  <CPU$REF> = <EXPR>
     (change an 2920 register or flag)
        Ex:   OVE = 0
(2)  <SM$REF> = <EXPR>
     (change the value of a SM2920 system variable)
        Ex:   SIZE = 188
(3)  <SYMBOLIC$REF> = <EXPR>
     (change the value of a symbol)
        Ex:   .PATCH = 101
(4)  RAM <ADDRESS$PARTITION> = <RAM$CHANGE$EXPR> [,<RAM$CHANGE$EXPR>]*
     (change 2920 data memory.  See RAM for more.)
        Ex:   RAM .TEMP = 0
(5)  ROM <ADDRESS$PARTITION> = <ROM$CHANGE$EXPR>
     (change 2920 code memory.  See ROM for more.)
        Ex:   ROM .NEXT = LDA DAR,KPO
See also BREAKPOINT,INPUT,QUALIFIER,SYMBOL,TRACE

<CONDITION> - A criterion composed of <COMPARISON>s connected with
ANDs and ORs.  AND has precedence over OR.
          <COMPARISON> [ AND/OR <COMPARISON>]*
   Ex:  PC=0
        TIME-TQUAL>0.0005
        PC=0 AND T MOD 0.0005 < TPROG OR OVF=1 OR DAR<>.OLD$DAR$VALUE
<COMPARISON>:
          <EXPR> <RELATION> <EXPR>
<RELATION>:
   <  =  >  <=  >=  <>

CONSOLE - A command which allows or suppresses trace data display
to the console during simulation.  This allows simulation to run
faster.  The trace data records are always accumulated in the trace
buffer whenever the QUALIFIER is satisfied.  If a LIST file is in
effect, trace data is filed regardless of the CONSOLE setting.
(1)   CONSOLE ON
      (Normal mode - trace data is displayed on the console device
      as it is collected)
(2)   CONSOLE OFF
      (Faster simulation mode - trace data display to console device
      is suppressed)
See LIST.

<CONSTANT> - A keyword which has a fixed numeric value.
ONE      The 2920 version of 1.0 i.e. 1.0-1/2**24
PI       3.14159D
TPI      Twice PI
HPI      Half of PI

<CPU$REF> - A keyword reference to an 2920 register or flag.
Each may be displayed by simply typing the keyword.  Those that are
not read only may be changed by typing the keyword followed by "="
and an <EXPR>.
PC               Program counter (0 to 191)
CY               Carry (0 or 1)
OVF              Overflow (0 or 1) (read only)
OVE              Overflow logic enabled (0 or 1)
DAR              Digital/Analog register (RAM 40)
IN               Sampled and held input signal in volts
OUT0,OUT1,OUT2,
OUT3,OUT4,OUT5,
OUT6,OUT7        Analog outputs in volts (read only)

<INPUT> - Refers to one of the simulated input signals IN0, IN1, IN2,
or IN3.  These may be specified as conditional expressions and are
evaluated during simulation whenever a corresponding input io-code is
encountered.
(1)   <INPUT>
      (Display the function which specifies the input signal and its
      current value)
         Ex:  IN2
              IN3
(2)   <INPUT> = [ [ IF <CONDITION> THEN <EXPR> ]* ELSE ] <EXPR>
      (Defines an input signal as a function)
          Ex:  IN0 = SIN(4400*TPI*T)
               IN1 = IF T<0.001 THEN SAW(10000*T) ELSE 0

DEFINE - Command to define a symbol:
(1)  DEFINE .<SYMBOL$NAME> = <EXPR>
     (add <SYMBOL$NAME> to end of symbol table and assign it a value.
     See also SYMBOL)
        Ex:  DEFINE .LABELX = 123
```

```
<DISPLAY> commands:
(1)  <CPU$REF>
     (display 2920 register, port, or pin.  Ex: DAR)
(2)  <SM$REF>
     (display SM2920 variable.  Ex: TIME)
(3)  <SYMBOLIC$REF>
     (display the value of a symbol.  Ex:  .ALPHA)
(4)  RAM <ADDRESS$PARTITION>
     (display 2920 data memory)
        Ex:   RAM .TEMP
(5)  ROM <ADDRESS$PARTITION>
     (display disassembled 2920 code memory)
        Ex:   ROM  .STAGE1 TO .STAGE2-1
See also BREAKPOINT,INPUT,QUALIFIER,TRACE,SYMBOL

EVALUATE - Command to evaluate an expression.  The value is displayed
in three bases (binary, decimal, hexadecimal).
              EVALUATE <EXPR> [SYMBOLICALLY]
                 Ex:   EVALUATE SQR(<RAM 0)*(RAM 0)+(RAM 1)*(RAM 1))
When the keyword "SYMBOLICALLY" is entered, then the value of the
expression is displayed as a symbol or as a symbol plus offset
if there is a symbol in the symbol table whose value is less than or
equal to the value of the expression.

EXIT - Command to exit the debugging session and return control to
ISIS-II.
             EXIT

<EXPR> - A numeric value that can be expressed in one of these forms:
(1)  <ADDRESS>
     (See <ADDRESS> for details)
        Ex:   SIN(1000*TPI*T)
              14
              .ARRAY+3
(2)  RAM <ADDRESS>
     (The value of <EXPR> is the contents of the data memory location.)
        Ex:   RAM .COUNTR

<FUNCTION> - A keyword which invokes a predefined function of its
argument when appearing in a <PRIMARY>, <ADDRESS>, or <EXPR>.
SIN      Sine function
COS      Cosine function
EXP      Exponentiation of e (2.71828)
LOG      Natural logarithm (inverse of EXP)
SQR      Square root
SAW      Saw wave version of sine
SQW      Square wave version of sine
ABS      Absolute value
See <ADDRESS>,<PRIMARY>.

GRAPH - A command to enable or disable graphic display of trace data.
The width of the trace display in either format is controlled by the
value of the <SM$REF> XSIZE.
(1)   GRAPH ON
      (Trace data is displayed in graphic format.  If first trace item
      is TIME it is used to label vertical axis;  otherwise the vertical
      axis is labeled incrementally.  The digits "0", "1", ... are used
      to mark the values of the first, second, ... trace items.
      An asterisk "*" is displayed when two or more digits overlap.
(2)   GRAPH OFF
      (Trace data is displayed in tabular format.  This is the default.)

HELP - Command to display a summary of the syntax of a command
or component of a command.
   (1)  HELP
        (display a list of all items for which there is help)
   (2)  HELP <IDENTIFIER>
        (display summary of specified item.  <IDENTIFIER> may contain
        dollar signs, but not angle brackets.)
           Ex:   HELP BREAKPOINT
   (3)  HELP *
        (display the summaries of all items)
The following notation is used in the help summaries:
    [A]     means A is optional.
    [A]N    means A is optional and may be repeated a maximum of N times.
    [A]*    means A is optional and may be repeated any number of times.
    A/B     means either A or B may be used.
    <A>     means there is also a help summary for A.
```

```
<IDENTIFIER> - A sequence of one or more of the following characters:
        ABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890?_
The first character can not be a decimal digit and only the first 31
characters are significant.  Dollar signs are allowed but are ignored.
         Ex:   SUM$0   SAVE$DAR   HELP

<INSTRUCTION> - A standard 2920 assembler instruction.  Shift-code
and io-code are optional.  ALU portion is optional too.
An operand may be a <NUMERIC$CONSTANT>, <SYMBOLIC$REF>,
DAR, KP0, KP1, KP2, KP3, KP4, KP5, KP6, KP7, KM1, KM2, KM3, KM4, KM5,
KM6, or KM7.
      Examples:  NOP
                 LDA 0,DAR,R6

LIST - Command which sends a copy of all console input and output
from the debugging session to an ISIS-II file.
              LIST <PATHNAME>
                 Ex:   LIST :F2:MYTEST.FIL
To close the LIST file (i.e., to terminate copying), type "LIST :CO:".
See CONSOLE for related commands.

LOAD - Command to load object code and/or its symbols.  The symbols are
added to the end of the SM2920 symbol table.
(1)  LOAD <PATHNAME>
     (load the object code and its symbols)
        Ex:   LOAD :F2:MYPROG.OBJ
(2)  LOAD <PATHNAME> NOSYMBOL
     (load only the object code)
(3)  LOAD <PATHNAME> NOCODE
     (load only the symbols)

MOVE - Command to move the trace buffer pointer   The pointer is
moved the specified number of records.
(1)  MOVE [ [+] <PRIMARY> ]
     (move toward newest trace.  MOVE without arguments is the same as
     "MOVE +1".)
        Ex:   MOVE 15
(2)  MOVE - <PRIMARY>
     (move toward oldest trace)
        Ex:   MOVE -50

NEWEST - Command to move the trace buffer pointer to just past the
newest trace in the buffer.
          NEWEST

<NUMERIC$CONSTANT> - A sequence of numeric characters (digits)
optionally including a period (".") followed by an optional suffix to
specify the constant's base.  If no suffix is specified, then the
constant is evaluated with the default suffix (see SUFFIX).  Dollar
signs may be used to improve readability, but are ignored.
   Examples:
      0.1001$1011B      (binary)
      .999D             (decimal)
      1FA9H             (hexadecimal)
      1.5               (default suffix)

OLDEST - Command to move the trace buffer pointer to the oldest trace
in the buffer.
          OLDEST

<PATHNAME> - The name of an ISIS-II file:
      Ex:   :F1:MYPROG
            :F2:TEST.001
            :HR:

<PRIMARY> -  A numeric value.  It has the following forms:
   <NUMERIC$CONSTANT>     (Ex:  1979, 0.1011B, 0.FFFCH, 3.14159D)
   <SYMBOLIC$REF>         (Ex:  .X1, .STAGE$2)
   <CPU$REF>              (Ex:  PC, DAR, CY)
   <SM$REF>               (Ex:  COUNT,BUFFERSIZE,TIME)
   <INPUT>                (Ex:  IN0, IN3)
   <CONSTANT>             (Ex:  PI, HPI, TPI, ONE)
   <FUNCTION> (<EXPR>)    (Ex:  SIN(T), SAW(1000*TPI*TIME))
   (<EXPR>)               (Ex:  (RAM .X1), (.FRED+14D))

PRINT - Command to display the trace buffer.  The buffer is displayed
by records.
(1)  PRINT [ [+] <PRIMARY> ]
     (Start at current trace pointer and move toward newer
     trace.  Trace pointer is left at one past last item printed.
     PRINT without arguments is the same as "PRINT +1".)
        Ex:   PRINT 15
```

```
(2)  PRINT - <PRIMARY>
     (Back up the trace pointer and then display forward.  Trace
     pointer is left at its original position.)
        Ex:   PRINT -15
(3)  PRINT ALL
     (Display the entire buffer, starting with the oldest trace.  The
     trace pointer is left at one past the newest trace.)

QUALIFIER -  May specify a condition for collecting trace.
(1)  QUALIFIER
     (display QUALIFIER)
(2)  QUALIFIER = <CONDITION>
     (specify QUALIFIER)
        Ex:  QUALIFIER = PC=0 AND T-TQUAL>0.0003

RAM - Command and keyword used to reference 2920 data memory:
   (1)  RAM <ADDRESS> [TO/LENGTH <ADDRESS>]
        (Display one or more locations)
           Ex:   RAM .ARRAYB LENGTH 12
   (2)  RAM <ADDRESS> = <RAM$CHANGE$EXPR> [,<RAM$CHANGE$EXPR>]*
        (Change one or more locations, starting at <ADDRESS>)
           Ex:   RAM .TEMP  =  0,ONE,1/2
                 RAM .ARRAY  =  RAM .TABLEX LENGTH 10
   (3)  RAM <ADDRESS> TO/LENGTH <ADDRESS>
                          = <RAM$CHANGE$EXPR> [,<RAM$CHANGE$EXPR>]*
        (Change several locations and perform range-checking or
        repetition.  If the data requires more memory than the size of
        the address range, then an error occurs.  If the data requires
        less memory, then the data is repeated until the range is
        filled.)
           Ex:   RAM .FLAGA TO .FLAGF  =  0
<RAM$CHANGE$EXPR>:
   (1)  <EXPR>
   (2)  RAM <ADDRESS> [TO/LENGTH <ADDRESS>]
        (used to move memory contents)

REMOVE - Command to delete a symbol from the symbol table.
(1)  REMOVE <SYMBOLIC$REF> [,<SYMBOLIC$REF>]*
     (remove one or more symbols from the symbol table)
        Ex:   REMOVE .LABELX
              REMOVE .MAIN,.LOOP
(2)  REMOVE SYMBOL
     (remove all symbols from the symbol table)

ROM - Command and keyword used to reference 2920 program memory:
   (1)  ROM <ADDRESS> [TO/LENGTH <ADDRESS>]
        (Display one or more locations as mnemonic instructions)
           Ex:   ROM 0 LENGTH 20
   (2)  ROM <ADDRESS> = <ROM$CHANGE$EXPR>
        (Change one or more locations, starting at <ADDRESS>)
           Ex:   ROM 3 = ADD .X0,KP4,R7,OUT0
                 ROM 100 = ROM 0 LENGTH 50
   (3)  ROM <ADDRESS> TO/LENGTH <ADDRESS>
                          = <ROM$CHANGE$EXPR>
        (Change several locations and perform range-checking or
        repetition.  If the data requires more memory than the size of
        the address range, then an error occurs.  If the data requires
        less memory, then the data is repeated until the range is
        filled.)
           Ex:   ROM .STAGE$1 TO .STAGE$2-1 = NOP
<ROM$CHANGE$EXPR>:
   (1)  <INSTRUCTION>
   (2)  ROM <ADDRESS> [TO/LENGTH <ADDRESS>]
        (used to move memory contents)

SAVE - Command to save code memory and/or symbols in an ISIS-II
hex format object file.
(1)  SAVE <PATHNAME>
     (save the last partition loaded and the symbol table)
        Ex:   SAVE :F1:NEWPRG
(2)  SAVE <PATHNAME> NOSYMBOL
     (save only the last partition loaded)
(3)  SAVE <PATHNAME> NOCODE
     (save only the symbol table)
(4)  SAVE <PATHNAME> <ADDRESS$PARTITION> [NOSYMBOL]
     (save specified part of code memory)
        Ex:   SAVE :F1:MYPROG 0 TO 191
              SAVE :HR:  .START LENGTH 20  NOSYMBOL
```

```
SIMULATE - Command to start simulation:
      SIMULATE [FROM 0] [<BREAK$COND>]
   If FROM 0 is specified, TIME and PC are set to 0, and the
   trace buffer is emptied before entering simulation.
   The BREAKPOINT contains the default <BREAK$COND>.  If
   <BREAK$COND> is specified, it is loaded into the BREAKPOINT.
<BREAK$COND>:
   (1)  FOREVER
        (BREAKPOINT becomes NEVER)
   (2)  ONCE
        (BREAKPOINT becomes ALWAYS)
   (3)  TILL <CONDITION>
        (BREAKPOINT becomes <CONDITION>)
          Ex: TILL PC=0 AND T>0.005
              TILL (RAM 0) MASK 0.0001B <>0 OR OVF=1

<SM$REF> - A keyword reference to an SM2920 system variable.  Each may
be displayed by simply typing the keyword.  Those which are not read
only may be changed by typing the keyword followed by "=" and an <EXPR>.
TIME          Simulated elapsed time (read only but can be zeroed
              - see SIMULATE)
TQUAL         Time when QUALIFIER last matched (and trace data was
              collected) (read only)
TPROG         Time for one pass through program
SIZE          Number of instructions in program
TINST         Time for execution of one instruction i.e. TPROG/SIZE
              (read only)
VREF          Reference voltage
BUFFERSIZE    Number of records of trace data in trace buffer
              (read only)
XSIZE         Width of graphic or tabular format trace display

SUFFIX - Refers to the default base of numeric constants that are input
from the console and have no explicit suffix.  Initially decimal.
See BASE for default display base.
   (1)  SUFFIX
        (display the default suffix)
   (2)  SUFFIX = <SUFFIX$TYPE>
        (change default suffix)
<SUFFIX$TYPE>:
   B      (Binary)
   D      (Decimal)
   H      (Hexadecimal)

SYMBOL - Refers to a symbol in the symbol table:
   (1)  <SYMBOLIC$REF>
        (display the value of a symbol)
           Ex:   .STAGE$1
                 .SUM
   (2)  SYMBOL
        (display the entire symbol table)
   (3)  DEFINE .<SYMBOL$NAME> = <EXPR>
        (add <SYMBOL$NAME> to end of symbol table and assign it a value)
           Ex:   DEFINE .OLD$DAR$VALUE = .13FH
   (4)  <SYMBOLIC$REF> = <EXPR>
        (change the value of a symbol)
           Ex:   .LABELX = 5
                 .OLD$DAR$VALUE = ONE/16
   (5)  REMOVE <SYMBOLIC$REF> [,<SYMBOLIC$REF>]*
        (remove one or more symbols from the symbol table)
           Ex:   REMOVE .SAVEIT
   (6)  REMOVE SYMBOL
        (remove all symbols from the symbol table)
<SYMBOL$NAME> - An <IDENTIFIER>.

<SYMBOLIC$REF> - A reference to a symbol in the symbol table.
See SYMBOL for related commands.
  (1)  .<SYMBOL$NAME>
       (Access first symbol in table with specified symbol name.)
          Ex:   .OLD$IN$VALUE

TRACE - Command to specify items to be traced during simulation
whenever the QUALIFIER is satisfied.
(1)  TRACE = <EXPR> [, <EXPR>]*
       Ex:   TRACE = TIME, OUT0, RAM 0, OUT0-SIN(TIME)
Related commands:  OLDEST, NEWEST, PRINT, MOVE
See also QUALIFIER and CONSOLE for trace data collection.
```

APPENDIX B
KEYWORDS AND SYNONYMS

This appendix gives all keywords which appear in this language, and their synonyms. Characters after the third are in lower case, since only the first three characters are checked.

Keyword	Synonym	Keyword	Synonym
ABS		OLDest	
ALL		ON	
AND		ONCe	
B		OR	
BASe		OUT0	
BREakpoint	B	OUT1	
BUFfersize		OUT2	
CONsole	C	OUT3	
COS		OUT4	
COUnt		OUT5	
CY		OUT6	
D		OUT7	
DAR		OVE	
DEFine		OVF	
ELSe		PC	
EVAluate	E	PI	
EXIt		PRInt	P
EXP		QUAlifier	Q
FORever.		RAM	
FROm	F	REMove	
GRAphics		ROM	
H		SAVe	
HELp	H	SAW	
HPI		SIMulate	S
IF		SIN	
IN		SQR	
IN0		SQW	
IN1		SUFfix	
IN2		SYMbol	
IN3		SYMbolically	
LENgth		TCLk	
LISt		THEn	
LOAd		TILI	T
LOG		TIMe	T
MASk		TO	
MOD		TPI	
MOVe	M	TPRog	
NEWest		TQUal	
NOCode		TRAce	
NOSymbol		VREf	
OFF		XSIze	

APPENDIX C
SYNTAX SUMMARY

This summary is intended as a definitive description of all legal combinations of commands, keywords, and options. To do so concisely requires a notation and an explanation of how to read it.

Some words or phrases are used in a special way, either to communicate an idea in English or to name a whole class of items, any of which may be used wherever this name appears in a command description in this summary. These words or phrases are enclosed in angle brackets <like this> .

The special notation ::= means "is defined as." A simple example is

<radix> ::= B | D | H

This should be read: "Radix is defined as the character 'B' or 'D' or 'H.'" This introduces two further aspects of the defining notation: to the right of the defining ::=, upper case characters or words are to be used exactly as written, whereas the angle-bracketed items, in lower case, indicate any member of the class named.

Command Summary

<simple comnd> ::=<display input comnd>
| <specify input comnd>
| <display breakpoint comnd>
| <set breakpoint comnd>
| <display qualifier comnd>
| <set qualifier comnd>
| <display trace selection comnd>
| <set trace selection comnd>
| <simulate comnd>
| <oldest comnd>
| <newest comnd>
| <print comnd>
| <move comnd>
| <display comnd>
| <change comnd>
| <define comnd>
| <display symbols comnd>
| <remove symbols comnd>
| <suffix comnd>
| <base comnd>
| <evaluate comnd>
| <list comnd>
| <console comnd>
| <help comnd>
| <exit comnd>
| < graphics comnd>

Expressions

<exp> ::=<ram reference>
| <address>
<address> ::=<arith term> [MASK <arith term>]*
<arith term> ::=<term> [<plus op> <term>]*
<plus op> ::=+|–

```
<term> ::=<factor> [<mult op> <factor>]*
<mult op> ::= * | / | MOD
<factor> ::=[<plus op>] <primary>
<primary> ::=(<exp>)
    | <function> (<exp>)
    | ONE | PI | TPI | HPI
    | <numeric constant>
    | <symbolic ref>
    | <keyword reference>
<function> ::= SIN | COS | EXP | LOG | SQR | SAW | SQW | ABS
<numeric constant> ::=<digit> [<digit>]* [<radix>]
    | [<digit>]* . <digit> [<digit>]* [<radix>]
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
    | A | B | C | D | E | F
<radix> ::= B | D | H
<symbolic ref> ::=<symbol>
<symbol> ::=. <identifier>
<keyword reference> ::=<2920 processor reference>
    | <SM-2920 reference>
<partition> ::=<address> [TO <address>]
    | <address> LENGTH <address>
```

Keyword Operators

```
<2920 processor reference> ::= IN0 | IN1 | IN2 | IN3 | OUT0 | OUT1 |
                               OUT2 | OUT3 | OUT4 | OUT5 | OUT6 |
                               OUT7 | IN | DAR | PC | CY | OVF | OVE
<SM-2920 reference> ::= TIME | TQUAL | COUNT | BUFFERSIZE | TINST |
                        TPROG | VREF | XSIZE
<ram reference> ::= RAM <address>
```

Simulation Controls and Commands

```
<display input comnd> ::=<input keyword reference>
<specify input comnd> ::=<input keyword reference> =<conditional exp>
<input keyword reference> ::= IN0 | IN1 | IN2 | IN3
<conditional exp> ::= [IF <comparison list> THEN <exp> ELSE]* <exp>
<comparison list> ::=<comparison> [<and/or> <comparison>]*
<comparison> ::=<exp> <relation> <exp>
<relation> ::=< | <= | =| >=|>
<and/or> ::= AND | OR
<display breakpoint comnd> ::= BREAKPOINT
<display qualifier comnd> ::= QUALIFIER
<set breakpoint comnd> ::= BREAKPOINT =<break/qual condition>
<set qualifier comnd> ::= QUALIFIER =<break/qual condition>
<break/qual condition> ::=<comparison list>
    | ALWAYS
    | NEVER
<display trace selection comnd> ::= TRACE
<set trace selection comnd> ::= TRACE =<trace item> [, <trace item>]
<trace item> ::=<exp>
<simulate comnd> ::= SIMULATE [FROM 0] [<break>]
<break> ::= TILL <comparison list> | FOREVER | ONCE
<oldest comnd> ::= OLDEST
<newest comnd> ::= NEWEST
<print comnd> ::= PRINT [[<plus op>] <primary>]
    | PRINT ALL
<move comnd> ::= MOVE [[<plus op>] <primary>]
<graphics comnd>::= GRAPHICS ON
    | GRAPHICS OFF
```

Interrogation and Utility Commands

```
<display comnd> ::= <keyword reference>
    | <symbolic reference>
    | RAM <partition>
    | ROM <partition>
<change comnd> ::= <keyword reference> = <exp>
    | <symbolic reference> = <exp>
    | RAM <partition> = <ram change exp> [, <ram change exp>]*
    | ROM <partition> = <rom change exp>
<ram change exp> ::= RAM <partition>
    | <exp>
<rom change exp> ::= ROM <partition>
    | <mnemonic instruction>
<define comnd> ::= DEFINE <symbol> = <exp>
<display symbols comnd> ::= SYMBOL
<remove symbols comnd> ::= REMOVE <symbolic ref list>
    | REMOVE SYMBOL
<symbolic ref list> ::= <symbolic ref> [, <symbolic ref>]*
<load comnd> ::= LOAD <path name> {NOCODE    }
                                  {NOSYMBOL}
<save comnd> ::= SAVE <path name> {NOCODE | <partition>}
                                  {NOSYMBOL           }
<suffix comnd> ::= SUFFIX [=<radix>]
<base comnd> ::= BASE [=<radix>]
<evaluate comnd> ::= EVALUATE <exp> [SYMBOLICALLY]
<list comnd> ::= LIST <path name>
<console comnd> ::= CONSOLE <on/off>
<on/off> ::= ON | OFF
<help comnd> ::= HELP [<help request>]
<help request> ::= <help item>
    | *
<help item> ::= <identifier>
<exit comnd> ::= EXIT
```

APPENDIX D
GETTING STARTED

Software Installation Procedure

The two modules of system software that support user prototype programs are the Intel Systems Implementation Supervisor (ISIS-II) and the SM2920 software module.

ISIS-II and SM2920 Diskette

The Intellec system uses the diskette hardware and ISIS to provide a powerful, convenient microcomputer development tool. ISIS interfaces to the diskette hardware and to any other standard peripheral device. You communicate with ISIS-II by entering commands on the system console or by embedding system calls in programs that will run in an ISIS environment. ISIS enables rapid storage and retrieval of files on diskette.

A diskette file containing 2920 assembly language source code is translated by the assembler, producing a hexadecimal format object file (absolute addresses) on diskette and a list file on the line printer or terminal output device, or on diskette. The main purpose of ISIS during setup and simulation is to provide the I/O interface to the console and files on diskette. To begin an SM2920 debugging session, you must first boot ISIS, next load the SM2920 software module, then load your object program.

Loading ISIS and the SM2920 Software Module

After installing all hardware and turning on power to the console, disk drives, and Intellec Series I, the following steps will load ISIS and SM2920. (64K is required; the full procedure and notes are in Chapter 2 of the *ISIS-II System User's Guide*):

a. Place system diskette in drive 0 and close drive door.

b. Press top half of boot switch on Intellec front panel.

c. Press top half of reset switch on that panel.

d. After interrupt light 2 goes on, press space bar.

e. After the light goes off, press bottom half of boot switch.

f. After receiving the ISIS sign-on message and prompt

```
—ISIS-II Vn.n
```

you type:

```
—SM2920
```

If the diskette containing the SM2920 module is not a system diskette, mount it in drive 1 instead of drive 0 and boot ISIS from a separate system diskette in drive 0. Then, at this point, rename one of the files to SM2920 as described on page 1-4 and type :F1:SM2920 (to load the SM2920 module from drive 1) followed by a carriage return. The SM2920 module will sign on and you are ready to continue.

If ISIS fails to sign on, recheck that all boards and cables are correctly installed and firmly seated, then perform the above procedure again.

If you have an Intellec Series II, there are slight variations in the above procedure. This is what you must do:

a. Turn on power to Intellec Series II and to disk drives. Press square white on/off button in the lower right-hand corner of front panel. It is a two-position switch; it lights up if you have pressed it correctly.

b. Note that this message appears:

 SERIES II, MONITOR, Vn.n

c. Place system diskette in drive 0 and close door.

d. Press reset button that is to left of the on/off button. Note that drive light is on to indicate that information is being accessed.

e. After receiving the ISIS sign-on message and prompt

 —ISIS-II Vn.n

 you type:

 —SM2920

 If the diskette containing the SM2920 module is not a system diskette, mount it in drive 1 instead of drive 0 and boot ISIS from a separate system diskette in drive 0. Then, at this point, type :F1:SM2920 (to load the SM2920 module from drive 1) followed by a carriage return. The SM2920 module will sign on and you are ready to continue.

If ISIS fails to sign on, recheck that all boards and cables are correctly installed and firmly seated, then perform the above procedure again.

APPENDIX E
ERROR MESSAGES

Error conditions encountered by the SM2920 module cause a numbered error message to print on your console.

Since commands are read on a line-by-line basis, the Simulator will not flag any error until after an entire line has been entered. When the first command error is found, command processing stops and the offending line has no further effect on any internal variables.

For example, the command

```
RAM 1 TO 4 = .02,.04,.06,.08,.05
```

will stop with the message

```
ERR 97: EXCESSIVE DATA
```

but it will have filled the 4 target locations with .02, .04, .06, and .08 respectively.

Error numbers C0 to CF are warnings of conditions which are probably undesired.

The following list of error messages does not include those which can come directly from ISIS-II. These appear in a separate list after the Simulator's messages.

In some rare cases, the error number may be printed without its associated message, as in

```
ERR 80: ?
```

This means an Error 80 was detected but some other (probably unrelated) problem prevented the Simulator from printing the message.

ERR 70:	CANNOT SIMULATE	Simulation cannot be performed because an EOP occurs at an address not divisible by 4.
ERR 71:	ILLEGIBLE NUMBER	A floating point number input cannot be deciphered.
ERR 72:	HELP FILE MISSING	The help file SM2920.OVH is missing.
ERR 80:	SYNTAX ERROR	The token flagged is not one that is allowed in the current context.
ERR 81:	INVALID TOKEN	The token flagged is illegal because it does not follow the rules for a well-formed token. The line is ignored and you must re-enter your intended command. Check the correctness of the syntax and variable-names used.
ERR 83:	INAPPROPRIATE NUMBER	The value printed on the preceding line is not appropriate in the current context. Some contexts allow only certain numbers, e.g., PC cannot exceed 191. This error can also arise during a SAVE command if a symbol has a noninteger value or a value not between 0 and 64K.

ERR 84:	PARTITION BOUNDS ERROR	The partition values entered in a command are not correct. Either the left part of the partition is greater than the right part, or the values of the partition extremes are out of range in the current context. For example, in the SAVE command, the partition must use integer values between 0 and 191.
ERR 85:	ITEM ALREADY EXISTS	The symbol entered in a define command is currently defined in the symbol table. You may need to validate the current usage of this symbol in your program, or perhaps merely use a different spelling to maintain the distinction.
ERR 86:	ITEM DOES NOT EXIST	The item printed on the preceding line does not reside in the symbol table. It may have been removed in an earlier test session, prior to saving the code, or it may be in a change you haven't inserted yet. Possibly you've loaded the wrong version of your code.
ERR 90:	MEMORY OVERFLOW	Too many symbols have been loaded or defined.
ERR 91:	STACK OVERFLOW	The capacity of a statically allocated internal stack has been exceeded. This is probably due to an excessively complicated command, e.g., one with 20 parenthesis pairs. An example would be DEFINE .DAR$SAVED = (((((((((((((((DAR))))))))))))))
ERR 92:	COMMAND TOO LONG	Too complicated a command due to number of operators, most probably, as in DEFINE .TEMPFUNC = 1+8*9−7/44* out to 30 or more operators. Break it up into several smaller commands. If issued in response to a SIMULATE command, this error indicates that the combination of the breakpoint, the qualifier, and the trace specifications is too complicated.
ERR 94:	NON-CHANGEABLE ITEM	An attempt to alter a read-only item, e.g., OUT7.
ERR 95:	INVALID OBJECT FILE	File specified in a load command is not a valid object file.
ERR 97:	EXCESSIVE DATA	The amount of data supplied for a partition exceeded the size of the partition. The n cells in the partition were filled with the first n data items supplied, after which the data given was ignored.
ERR 9D:	LINE TOO LONG	Command line was longer than 122 characters.

ERR B8:	ASSEMBLY IMPOSSIBLE	A mnemonic instruction cannot be assembled, even though its syntax is reasonable, because the instruction does not exist.
ERR B9:	NO HELP AVAILABLE	Help has been requested for a help item which has no help message.
WARN C0:	EXTRA EOP	The 2920 program currently in memory has more than one EOP. The extra EOP is at the location specified in the line above the warning.
WARN C1:	EOP NOT MULT OF 4	The 2920 program currently in memory has an EOP at an address not divisible by 4. The offending EOP is at the location specified in the line above the warning.
WARN C2:	NO EOP	The 2920 program currently in memory has no EOP at all. This is actually a mild warning, as many 2920 programs will have no EOP.
WARN C3:	DAR USED TWICE	The 2920 program currently in memory contains an instruction which uses the DAR twice, for example LDA DAR,.FNT1,CVTS or SUB DAR,.FN2,CND1 The location of the offending instruction appears on the line preceding the warning. Since the 2920 itself acts unpredictably in this case, simulation cannot be accurate.
WARN C4:	OUT FOLLOWS DAR CHANGE	The 2920 program currently in memory contains a sequence of instructions in which an output iocode follows an instruction in which the DAR is changed. The location of the offending output iocode appears on the line preceding the warning. Since the 2920 itself acts unpredictably in this case, simulation cannot be accurate.
WARN C5:	CVT FOLLOWS IN OR DAR CHANGE	The 2920 program currently in memory contains a sequence of instructions in which a convert iocode follows an instruction which contains an input iocode or an instruction in which the DAR is changed. The location of the offending convert iocode appears on the line preceding the warning. Since the 2920 itself acts unpredictably in this case, simulation cannot be accurate.

WARN C6:	NUMBER NOT BETWEEN −1 AND +1	Sometime during the last command (possibly a simulation) an input signal specification exceeded the range −VREF to +VREF, or an attempt to change RAM contents to a value not between +1 and −1 was made. In either case, the limiting value closest to the offending value was selected and used.
WARN C8:	F.P. INVALID OPERAND	The program tried to use a value resulting from an underflow or overflow condition.
WARN C9:	F.P. OVERFLOW	A value larger than 10 times the largest allowable number occurred in some expression.
WARN CA:	F.P. UNDERFLOW	A value smaller than the smallest allowable number occurred. One example is 1/largest#.
WARN CB:	F.P. ZERO-DIVIDE	Dividing by zero was attempted.
WARN CC:	F.P. DOMAIN ERROR	One example would be the square root of a negative number.

The last five warnings are flagged during a command executed, possibly a simulation, due to an inappropriate action or result for a floating point operation. See the documentation for the FORTRAN floating point libraries for further details.

ERR E7:	ILLEGAL FILENAME	The filename specified does not conform to a well-formed ISIS filename. See *ISIS Manual* for valid formulation and device labels.
ERR E8:	ILLEGAL DEVICE	Illegal or unrecognized device in filename. An invalid device label was used, e.g., :D0: instead of :C0, or something unrelated such as :PQ:. See *ISIS Manual* for valid list.
ERR E9:	FILE OPEN FOR INPUT	Attempt to write to a file open for input, e.g., SAVE :CI:, a file predefined as console input.
ERR EF:	FILE ALREADY OPEN	Attempt to open a file that was already open.
ERR F0:	NO SUCH FILE	The file specified does not exist. Possibly a wrong or missing device label, as in typing :F2:FILE when you meant :F3:FILE, or a file missing due to forgetting to copy it onto a new disk.
ERR F1:	WRITE-PROTECTED FILE	The file named for output is write-protected and cannot be overwritten.
ERR F3:	CHECKSUM ERROR	A checksum error in a hex object file was encountered during loading.
ERR F6:	DISKETTE FILE REQUIRED	A file was referenced which needs a diskette.

ERR F9:	ILLEGAL ACCESS	Attempt to open a read-only file for the purpose of storing data (e.g., specifying :CI: as the list device) or to open a write-only file as a source of data (e.g., :LP in a load command).
ERR FA:	NO FILE NAME	No filename specified for a diskette file (e.g., no filename following :F2:).
ERR FH:	NULL FILE EXTENSION	An expected filename extension was not found (e.g., :F2:FILT).

APPENDIX F
LIST OF COMMANDS

Command	Brief Description of Use
HELP	Look at brief descriptions of any or all commands.
LOAD	Get your program from its file into memory where it can be simulated.
LIST	Set up where the record of this simulation session (or any part of it) is to go.
SUFFIX	Set up default radix for numeric input.
BASE	Set up what number base you want used when numbers are displayed, i.e., binary, decimal, or hexadecimal.
Display	Display is not a keyword. The display of addresses or contents of any symbol or register is achieved by simply typing the name of the element desired, and then a carriage-return.
EVALUATE	Evaluate numbers or expressions in all bases.
Change	Change is not a keyword. To alter symbolic constants or the contents of RAM or ROM locations, registers, or bits, the name is typed followed by an equal sign followed by the intended new-contents or new-value.
BREAKPOINT	Display or change the halting conditions to stop simulation.
QUALIFIER	Display or change the conditions allowing or inhibiting the collection of trace data during simulation.
SYMBOL	Display symbols currently in table.
DEFINE	Set up and assign values to new symbols.
REMOVE	Destroy specific (or all) symbol table entries.
INPUT	Define input signals as conditional functions of time and any other variable-names, user or processor.
CONSOLE	Optionally turn on console trace display during simulation.
TRACE	Display or alter the specific data items or expressions to be traced when trace is enabled.
GRAPHICS	Optionally turn on graphing of trace data instead of tabulation.
SIMULATE	Begin a simulation at zero or where the last simulation stopped or at an address you specify, optionally using prior halting conditions or specifying new ones.
OLDEST, NEWEST MOVE	Move pointer in trace buffer to earliest or latest record traced, or adjust buffer pointer to desired record before using PRINT command.
PRINT	Display trace buffer records.
RAM address	Change or display RAM.
ROM address	Change or display ROM.
SAVE	Save modified code.
EXIT	Ends this simulation session, terminates the Simulator, returns control to ISIS.

APPENDIX G
HEXADECIMAL OBJECT FILE FORMAT

All user programs loaded via the SM2920 module must conform to Intel's standard for hexademical object files, partly because the language translators generate only hexadecimal code.The hexadecimal object code file generated by the AS2920 assembler contains the contents of program memory which would result from loading the assembled source program. The code is formatted in hexadecimal bytes of data. The file contains the ASCII representation of the hexadecimal bytes of data. The object code itself is preceded by a symbol table. These two parts may be loaded or saved together or separately.

The symbol table is a series of records, terminated by a dollar sign. Each record contains three fields separated by one or more ASCII spaces:

- a number field (not used by SM2920)
- a label field containing the ASCII representation of a source program symbol, and
- an address field containing the hexadecimal address assigned to the symbol by the language translator.

The symbol table is terminated by a record whose first nonblank character is a dollar sign.

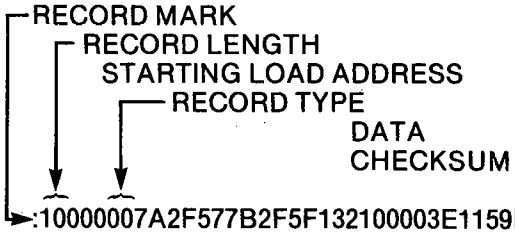
The object code generated by the language translator follows the symbol table. The symbol table has records. Each of these records or physical lines is six logical fields of varying length in characters or frames:

FIELD 0: RECORD MARK (FRAME 0 IS ALWAYS ':')
FIELD 1: RECORD LENGTH (FRAMES 1 AND 2)
FIELD 2: LOAD ADDRESS FIELD (FRAMES 3,4,5 AND 6)
FIELD 3: RECORD TYPE (FRAMES 7 AND 8)
FIELD 4: DATA (FRAMES 9 TO 9+2*[RECORD LENGTH]-1)
FIELD 5: CHECKSUM (FRAMES 'DATA FIELD' + AND 'DATA FIELD' +2)

For an example of the object file format see the sample program in Appendix A.

Object Code Output Format

The format of the object code is a series of records, each containing its record length, type, memory load address, checksum, and data. The figure following shows a typical output file in hexdecimal format.




```
:10000007A2F577B2F5F132100003E1159D21282
:1000000000E3E1F579174F7817477D176F7C176775
:0D002000F13DC20C00B77C1F577D1F5FC96A
:00000001
```

(Because record length equals 0 and record type equals 01, this record specifies end-of-file.)

INDEX

abbreviation
 list of command, Appendix B
 of command keywords, 1-5
abort a command, or a simulation, 1-5, 3-18
ABS, 3-20
addresses, Preface, 2-6, 2-7
 numeric, 1-1 thru 1-3, 2-4
 range, 3-10 thru 3-14
 starting, 1-1, 2-1, 3-18. *See also* set PC
 symbolic, 1-1, 1-2, 2-4, 3-14
ALL, 2-4. *See* PRINT
ALWAYS, 3-19, 3-21, 3-25
ampersand, 1-4, 3-3
analog input, 1-4, 4-1
AND, 1-3, 2-4, 3-19, 3-20, 3-25. *See also* set breakpoint, set qualifier
ASCII, 2-2. *See also* ISIS manual
assembling, 1-1
asterisk
 double, 3-3
 in HELP, 1-5
 prompt, 1-1, 3-3, 3-20
at-sign, 2-5

B. *See* breakpoint, binary, suffix
BASE, 3-8, 4-3
binary radix, 2-5
blanks
 as token separators, 3-3
 in SAVE, 3-5
"booting" the system, 1-1, Appendix D
braces, 2-9. *See* notation
brackets, 2-8. *See* notation
break
 point register, 1-2, 3-18
 displaying, 3-20
 prior contents last, 3-19
 setting, 1-2, 3-19
 -s in simulation, 1-3, 1-4, 4-1
BREAKPOINT, 1-1, 2-3, 3-17 thru 3-20, 4-1
buffer, 1-2, 3-21, 4-5
 pointer, 3-21, 3-22
BUFFERSIZE, 1-6, 3-24, 4-5

capacitor constants, 4-1, 4-2
carriage-return, 1-4, 3-3, 3-9, 3-10, 3-15, 3-20, 3-22
Carry flag (CY), 1-6
Change, Preface. *See also* set
 memory contents, 3-11
 processor and SM2920 referents, 3-9
 symbols, 3-16
characters, 2-2
 alphabetic, 2-2
 ASCII, 2-2
 numeric, 2-2
 special, 2-2
 strings, 2-1, 2-2
code, Preface
 debugged, 1-2
 memory, 1-1, 3-4
 object, 3-4
 program, 1-2
columns of trace display, 3-8, 3-22
comma
 between new-contents, 3-12
 between trace items, 3-22
 in lists, 3-3, 3-15
commands
 entering at console, 1-4, 3-3
 in progress, 1-5
 lines, 3-3
 list, Appendix F
 memory-content, 3-9
 notation, 2-8
 simulation control, 3-17
 structure, 2-1
 summarized, 3-2
 symbol table, 3-14
 TRACE control, 3-21
 utility, 3-4
comments, 1-4, 3-3, 4-3
concepts of simulation, testing, and debugging, 1-1
Conditions, 1-2, 1-3, 3-19, 3-20, 3-25. *See* expressions, relational. *See also* IF, breakpoint, qualifier and simulation commands
configuration, Preface
CONSOLE, 1-2, 3-21, 4-5
console, Preface
 command, 3-21
 entering commands, 3-3
 input, 3-5
constants
 floating point, 2-5
 HPI, 3-20
 legal range, 3-9, 3-13
 numeric, 2-5, 3-14
 ONE, 3-20
 PI, 3-20
 premultiplied, 4-2
 sample-and-hold capacitors', 4-1, 4-2
 TPI, 3-20
continuation lines, 3-3
COS, 2-7
COUNT, 1-6
cpu$ref 1, 1-5, Appendix A
CTRL P, 3-3
CTRL R, 3-3
CTRL X, 3-3
CY. *See* carry

D. *See* suffix, decimal
DAR, 1-6, 2-3, 3-9, 4-3. *See also* digital/analog register
data memory, Preface, 1-1
 addressed via names in symbol table, 2-4

DEBUG option of assembly, 1-2, 1-3, 2-5, 3-14
Debugging
 concepts, 1-1
 symbolic, 1-2
decimal radix, 2-5
DEFINE, 2-5, 3-15, 3-16
define
 input, Preface, 1-4, 3-20
 symbols, Preface, 1-2, 3-14 thru 3-16
device name, 3-4
digit, 2-5
digital/analog register (DAR), 1-6
directory, 3-4
disassembly, 1-3
diskette, 1-2, 3-4, Appendix D
display, Preface, 1-4
 2920 processor referents, 3-9 thru 3-14
 breakpoint, 3-20
 memory contents, 3-11
 qualifier, 3-26
 SM2920 referents, 3-9 thru 3-14
 symbols, 3-15
 TRACE data records, 3-23, 3-24
 TRACE items, 3-22
dollar sign, 3-3, 3-15
drive
 diskette, 3-4
dumps (to trace buffer) at fixed times, 4-2

editing, 3-3
ELSE, 3-20
EOP, 1-6
 effect on simulation, 4-1
equal sign. *See* change, set
error messages, Appendix E
ESCape key, 1-5, 3-3, 3-17 thru 3-19, 4-2
EVALUATE, 3-8, 4-3
EXIT, 1-5, 3-4, 3-5
EXP, 2-7, 3-20
exponentation, 3-20
expressions
 defined, 2-6, 2-7
 evaluator, 1-4, 3-8
 in parentheses, 2-6
 in partitions, 2-7
 relational, 1-3, 3-19, 3-25
extension filename, 3-4

filename, 3-4 thru 3-6
flags, Preface, 1-2
floating point constants, 2-5
FOREVER, 2-4, 2-8, 3-19
FORTRAN, 1-3
FROM, 1-3, 2-4
 clause, 2-1, 3-18
functions, Preface, 3-20
 built-in, 3-20
 numeric, 3-20
 COS, 3-20
 EXP, 3-20
 LOG, 3-20
 SAW, 3-20
 SIN, 3-20
 SQR, 3-20
 SQW, 3-20
 input, 1-4

Getting Started. *See* Appendices A, D and Chapter 4
GRAPHICS, for trace, 3-26

H. *See* suffix, hexadecimal
halting conditions, Preface, 1-4, 2-1, 2-4, 3-18. *See also* breakpoint
HELP, 1-5, Appendix A
help
 command, 1-5, Appendix A
 facility, 1-5
 messages, 1-4, Appendix A
HEX, 3-4
hexadecimal
 format, 1-3, Appendix G
 -files, 1-3
 number base, 2-5
 output example, 3-11
HPI, 2-7, 3-20

IF, 3-20, 3-21, 4-1
IN, 1-6, 3-20
IN0, IN1, IN2, IN3, 1-6, 3-20, 4-2, 4-4
In-Circuit Emulator-85, 1-4
input
 channel, 2-8
 functions, 1-2, 1-4, 3-20
 affect speed, 4-1, 4-2
 iocode, 4-1
 latest sampled (IN), 1-6
 line length, 1-4, 3-3
 radix. *See* SUFFIX
 sampling. *See* sampling
 simulation of, 1-4
 specification, 1-1, 3-17. *See also* input, functions, and IF
instruction
 in change commands, 3-11 thru 3-13
 number in program (SIZE), 1-6
 number of, simulated (COUNT), 1-6
INTELLEC, Preface
 manuals, Preface
 memory, 1-2
interrogation mode, 1-1, 1-2
I/O (input/output). *See* ports
iSBC-310 math board, Preface, 1-4
ISIS, Preface, 3-4, Appendix D
 disk files, Preface, 3-4
 editing, 3-3
 keyboard and I/O functions, Preface
 manuals, Preface
 prompt, 1-1
 rename command, 1-4
items. *See also* trace
 separator, 2-9

keyboard calculator, 1-4
Keywords, 2-2, Appendix B
 command, 2-3
 connector, 2-4
 function, 2-3
 mode, 2-3
 object, 2-4
 operational control, 2-4
 reference, 1-5, 1-6, 2-3

labels, names for ROM locations, 3-14
LENGTH, 3-7, 3-10
line
 continuation, 3-3
 feed, 1-4, 3-3
 input, 3-3
 length of input, 1-4
 printer, Preface
LIST, 3-4, 3-6, 3-21, 4-5
list of new contents, 3-11 thru 3-14
 too long, 3-14
 too short, 3-13
LOAD, 1-2, 2-5, 3-4, 3-5, 3-14, 3-18, 4-3
 -ing, Preface, 1-3, 3-14
LOG, 3-20

MASK, 2-6
match, 2-1
math board, Preface, 1-4
Memory. *See* entries under program, data
 content references, 3-9
 displaying, 1-2, 3-11
 setting, 1-2, 3-11 thru 3-13
Mnemonics, 2-1
MOD, 2-6, 4-2
Modes
 functional, 2-1
 of operation, 1-1, 1-2
MOVE, 3-21, 3-22, 3-24, 4-5

names
 for RAM locations. *See* variables
 for ROM locations, 3-14
 symbolic, 1-1, 1-2
NEVER, 3-18
NEWEST, 3-21, 3-24, 4-5
NOCODE, 3-5, 3-6
NOP, 4-1, 4-2
NOSYMBOL, 3-5, 3-6
notation, 2-8, A-3. *See also* syntax
number
 bases, 3-7. *See also* radix
 constants, 2-5
 range of values, 3-9
numeric
 constant, 2-5
 expression, 2-1

object
 code, 1-2, 1-3, 3-4
 file, 1-3
OFF, 2-4, 3-21. *See also* console
OLDEST, 2-4, 3-21, 3-24, 4-5
ON, 1-2, 2-4, 3-21. *See also* console
ONCE, 1-3, 2-4, 2-8, 3-19
ONE, 2-7, 3-20
operands, 2-6
operation modes, 1-1
operators, 2-6
 MASK, 2-6
 MOD, 2-6
 mult, 2-6
 plus, 2-6
 precedence, 2-6
 relational, 2-6
optional command controls, 2-8

OR, 2-1, 2-4, 3-19, 3-20, 3-25. *See also* set breakpoint, set qualifier
OUT0, OUT1, OUT2,
 OUT3, OUT4, OUT5,
 OUT6, OUT7, 1-6
output radix. *See* BASE
OVE. *See* overflow limiting enable bit
overflow
 flag bit (OVF), 2-1, 2-4
 limiting enable bit (OVE), 1-6
OVF, 2-1, 3-9, 3-20. *See also* overflow flag bit

parentheses, 2-6
Partitions, 2-7, 3-5, 3-6, 3-10 thru 3-14
patch, a program change, *See* 3-11 thru 3-14
pathname, 3-4 to 3-6
PC, *See* program counter
periods, 2-5, 4-3
PI, 2-7, 3-20
ports, 1-2
precedence
 of operators, 2-6
primary, 2-6, 2-7, 3-22 thru 3-25
PRINT, 2-4, 3-23, 3-24, 4-4
processor, Preface
 register, 1-1
 reference, Preface, 1-5, 2-3
 states, Preface
 variables, Preface
Program
 checkout, 1-1
 counter, 1-3, 1-6, 2-3, 3-9, 3-18, 3-19, 3-24, 4-1, 4-2
 loading, 1-2
 memory, Preface, 1-1, 1-3
 See also ROM
 object, Preface
 pass, 4-2
 size, 1-6
prompt character
 for assembler, 1-1, 1-2, 3-20
 for ISIS, 1-1, Appendix D
pseudoregisters, 1-2, 1-5
pulse, 3-21
Punctuation, 2-1
 ampersand, 1-4, 2-1
 carriage-return line feed, 1-4, 2-1
 semicolon, 1-4, 2-1

QUALIFIER, 1-2, 2-3, 4-4
 displaying, 3-21, 3-26
 setting, 2-4, 3-17, 3-25
 use, 1-3, 2-4, 4-2
question mark, 2-5

radix, 2-5, 3-7, 3-8
 binary (B)
 decimal (D)
 hexadecimal (H)
 See BASE, SUFFIX, EVALUATE
RAM, 1-1, 1-3, 2-1, 2-7, 3-9 thru 3-14, 3-20, 4-4
 address, 2-7
 See also data memory

read-only
 memory, 1-1
 system variables or registers, 1-4 thru 1-6
records
 limit on trace, 1-3
References
 processor (2920)
 displaying, 3-9 thru 3-14
 setting, 3-9
 Simulator (SM2920)
 displaying, 3-9 thru 3-14
 setting, 3-9
Registers, Preface, 3-9
relational
 expressions, 3-19
 operators, 2-6
REMOVE, 2-4, 3-15, 3-16
RENAME, 1-4
repeating list, 3-13
ROM, 1-1, 1-2, 2-1, 3-4, 3-9 thru 3-14, 3-18, 4-3
 addressed via labels in symbol table, 2-4
 See also program memory
RUBOUT, 3-3

Sampling, Preface, 4-1
 simulated in software, 1-4, 3-20
SAVE, 1-2, 2-7, 3-4 thru 3-6
 ing, Preface, 1-3
SAW, 2-7, 3-20, 4-6
semicolon, 1-4, 3-3
set
 breakpoint, 1-2, 3-19
 memory contents, 3-11 thru 3-13
 PC, 1-3, 3-9, 3-18
 qualifier, 1-2, 3-25
 2920 processor referents, 3-9
 SM2920 referents, 3-9
 symbols, 3-15, 3-16
 trace items, 1-2
SIMULATE, 1-2, 2-1, 3-17, 3-19, 4-4
Simulation
 breakpoints, 1-1, 3-18 thru 3-20
 command, 1-2, 2-1, 3-19
 concepts, 1-1
 control commands, 3-17
 example, 4-1
 general session, 1-1
 halting conditions, 2-1, 3-18
 See breakpoints
 mode, 1-1
 of each instruction, 4-1
 of EOP, 4-1
 performed in software, 1-4
 registers, 2-3
 simple with defaults, 3-18
 speed, Preface
 increasing, 4-1
 unpredictable, E-3
Simulator
 files, 1-4
 invoking, 1-4, Appendix D
 registers, Preface, 3-9
 references, 1-5
SIN, 2-7, 3-20, 4-1, 4-4
sine wave, 3-20, 4-2
SIZE, 1-6, 3-5, 4-1, 4-4
slash
 as division operator, 2-6
 in command syntax, 2-9
SM2920, 1-4, 2-3, 3-20, Appendix D. *See also* Simulator
SM2920.HRD, Preface, 1-4
SM2920.0VH, 1-4
SM2920.SFT, Preface, 1-4
sm$ref, 1-5, Appendix A
space, 1-4
SQR, 2-7, 3-20
SQW, 2-7, 3-20
starting location
 of partition, 2-7
 of simulation, 1-1, 1-3, 2-1, 3-18
SUBMIT, 3-3, 4-1
SUFFIX, 3-7, 4-3
SYMBOL, 2-4, 3-15, 3-16, 4-3
symbol
 syntax, 2-4, 3-15
 table, 1-2, 1-3, 2-4, 2-5, 3-3 thru 3-6
 commands, 3-14
 deletion, 3-6, 3-15
symbolic references, Preface, 1-1, 1-3, 2-5
 addressing, 2-4
 debugging, 1-2
SYMBOLICALLY, 3-8. *See also* EVALUATE
syntax, Chapter 2 and Appendix C
 notation, 2-8
 per command, Chapter 3
 symbol, 3-15

testing
 concepts, 1-1
 unattended, 3-3, 4-1
THEN, 3-20
TILL, 1-3, 2-1, 2-4
 clause, 2-1, 2-8, 3-17
TIME, (see time), 3-17, 3-19, 3-24
time
 elapsed (TIME), 1-2, 1-4, 1-5, 3-17, 3-19, 3-24, 4-1
 of last match on qualifier (TQUAL), 1-5, 3-20, 4-2
 per instruction (TINST), 1-5, 4-1
 per program pass (TPROG), 1-5, 4-1
TINST, 4-1, 4-4. *See also* time
TO, 2-7, 3-10
tokens, 2-1, 2-2
 special
 operators, 2-6
 punctuation, 2-6
TPI, 2-7, 3-20
TPROG , 4-1, 4-2, 4-4. *See also* time
TQUAL, 3-20, 4-2. *See also* time
TRACE, 1-1, 3-17, 3-21, 3-22, 4-4
trace
 buffer, 1-1, 1-2, 3-21, 4-5
 pointer, 1-2, 3-21, 3-22
 size, 1-6
 control commands, 3-21

data, 1-1 thru 1-3, 3-5, 3-8, 3-21, 3-23, 3-24
-ing, Preface, 1-1
at fixed times, 4-2
inhibit, 3-25
items, 1-2, 1-3, 3-17, 3-21
displaying, 3-22
setting, 3-22
qualifiers, 1-3, 3-25
records, 3-23
limit, 1-3, 3-17

underline, 2-5

user
created symbols, 2-4
named locations, Preface, 2-1
names and symbols, 2-4
utility commands, 3-4

variables, Preface, 1-1 thru 1-4, 3-14
legal range, 3-9, 3-13
system, 1-5, 3-20
voltage reference (VREF), 1-6, 3-17
VREF, 1-6

XSIZE, for trace, 1-6, 3-26

REQUEST FOR READER'S COMMENTS

The Microcomputer Division Technical Publications Department attempts to provide documents that meet the needs of all Intel product users. This form lets you participate directly in the documentation process.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. _____

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

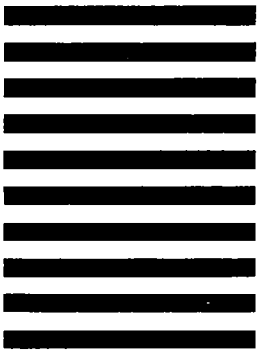
Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS ...

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.



NO POSTAGE
NECESSARY
IF MAILED
IN U.S.A.



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1040 SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
Attn: Technical Publications M/S 6-2000
3065 Bowers Avenue
Santa Clara, CA 95051



INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 987-8080

Printed in U.S.A.